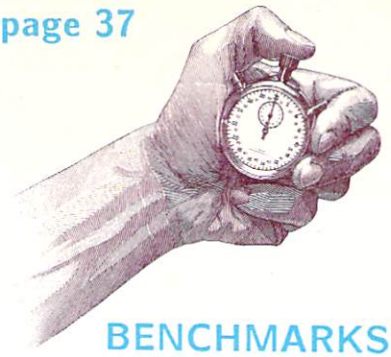
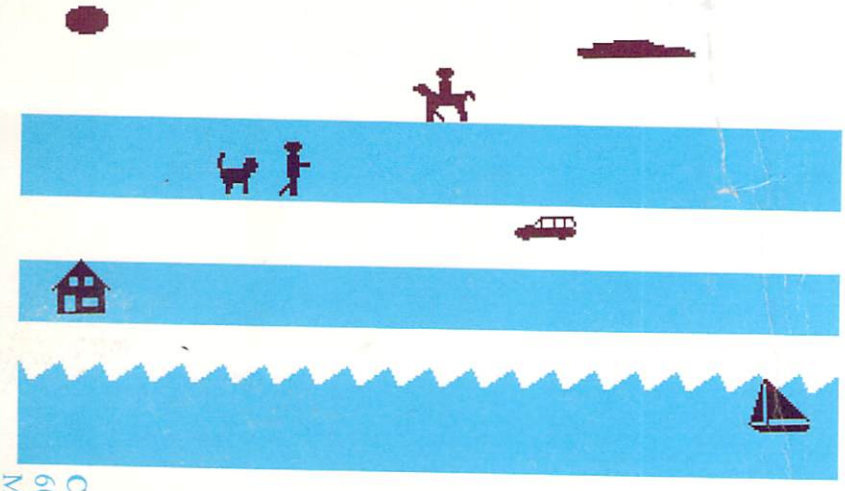
COMAL TODAY 12

page 37

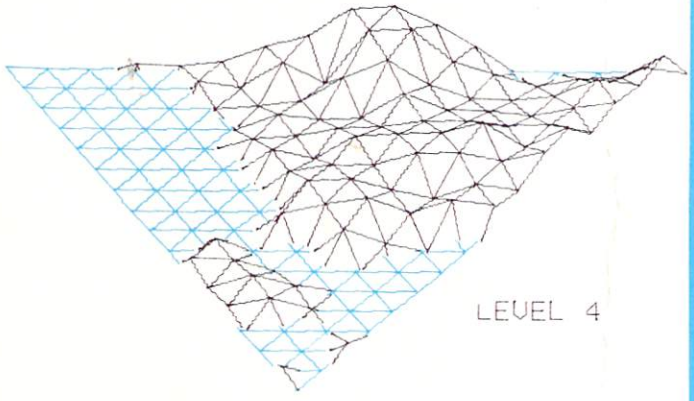


BENCHMARKS

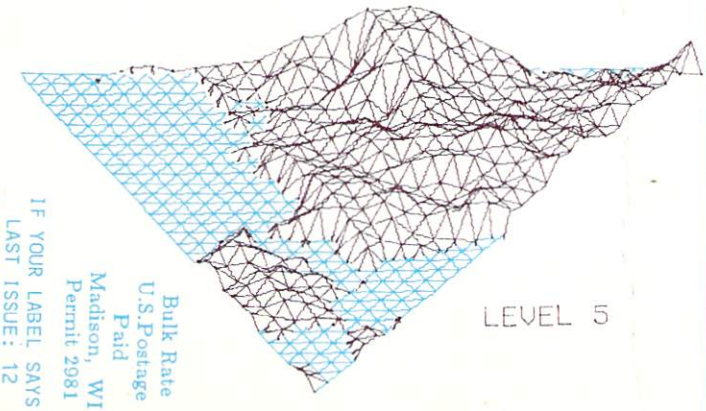
KELLY'S BEACH page 62



COMAL Today
6041 Monona Drive
Madison, WI 53716



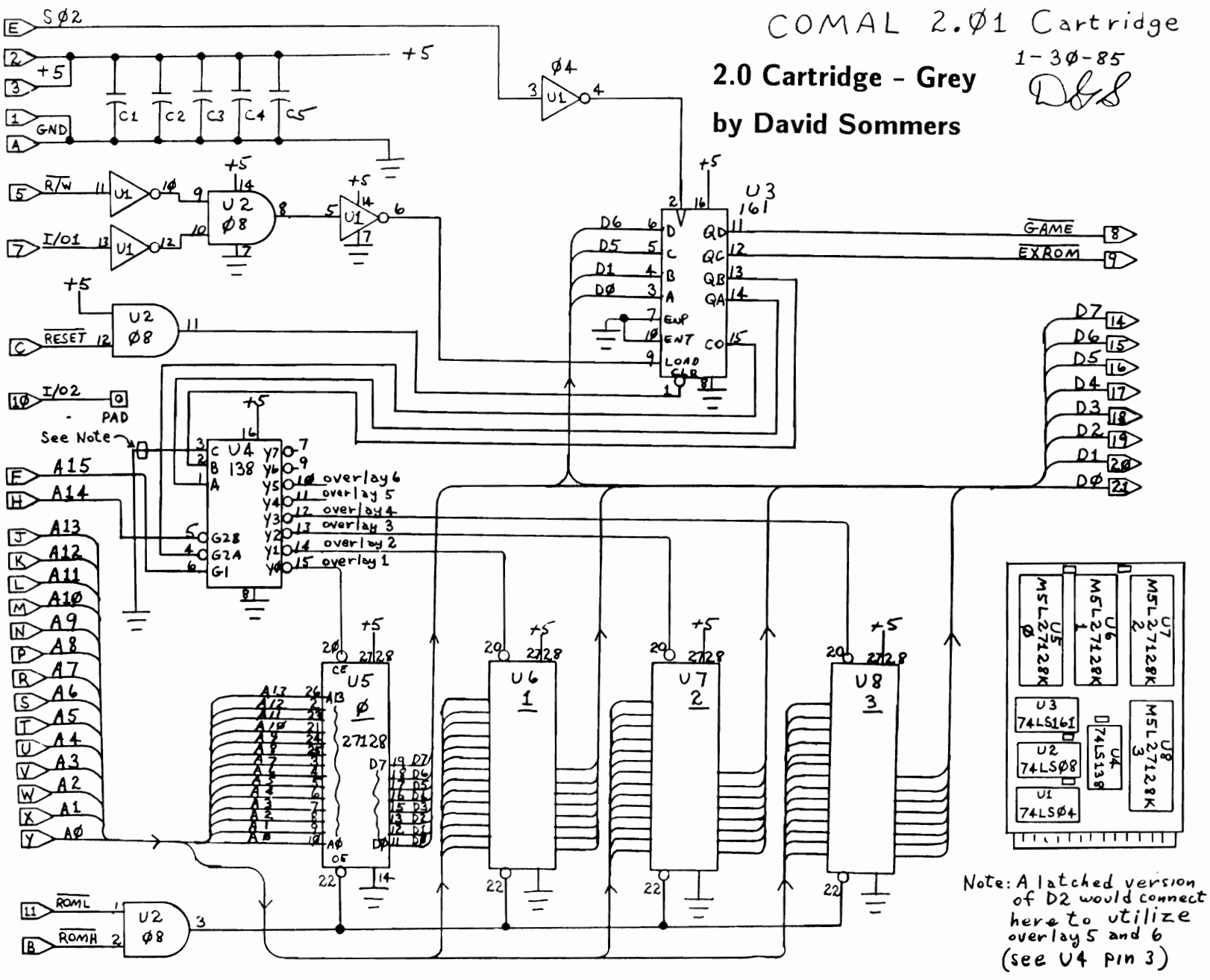
3D FRACTALS page 26



Bulk Rate
U.S.Postage
Paid
Madison, WI
Permit 2981

IF YOUR LABEL SAYS
LAST ISSUE: 12
YOU MUST RENEW NOW.
USE ORDER FORM INSIDE

Cartridge Schematic



COMAL Today Issue 12 April 24, 1986

PUBLISHER
COMAL Users Group,
U.S.A., Limited
6041 Monona Drive
Madison, WI 53716

EDITOR
Len Lindsay

ASSISTANTS
Denise Bernstein
Maria Lindsay
David Stidolph
Geoffrey Turney

ART
G Raymond Eddy

CONTRIBUTORS
Phyrne Bacon
B W Behling
Marcel Bokhorst
Ed Bolton
Ray Carter
Doug Colpitts
Captain COMAL

CONTRIBUTORS
Herbert Denaci
David Funk
Garrett Hughes
K Keller
Travis Lappe
J William Leary
Len Lindsay
Ed Matthews
Bob McCauley
Ralph McMullen
Daniel Parish
Kevin Quiggle
Joel Rea
Jay Renard
Terry Ricketts
D Shellenberg
David Sommers
William Staneski
David Stidolph
Scott Strool
Colin Thompson
Craig VanDegrift
Jim Ventola
Joe Visser
Lowell Zabel

From the Editor's Disk

by Len Lindsay

I must apologize for the poor printing quality last issue. We changed printers to one closer to us. Needless to say, we are using the original printer again. We think it is worth an hour drive to get better quality. We try to make *COMAL Today* better each issue. Readers wanted to know what version of COMAL applied to each article. We now use small icon pictures: a **disk** means C64 COMAL 0.14; a **cart** means 2.0; an **I** means IBM PC COMAL.

IMPORTANT News: AHOY Magazine is now including 400 blocks of COMAL programs on each of their monthly disks, including COMAL 0.14 and fast loader. If they get enough letters requesting it, they said they would begin printing COMAL articles! **Read their letter on page 72,** then send them a letter. Keep a copy of your letter. Next issue I'll explain how it can be valuable to you!

A decision needs to be made, and you can help. Our first issue was 28 pages without a cover. Our subscription price was based on that, and has not increased, even though we now have 80 pages plus a heavy cover. If we listed every program on a typical disk it would take over 100 pages even using small print and no articles! However, nearly half of our subscribers also get the disk and want us to print just the articles about the larger programs. We could increase the subscription price and print more pages in each issue. We do have plenty of material.

Here is a partial list of articles/ programs that are ready to publish: Wheel of Fortune, Zoo Match Game, Frustration Factors, Custom Directories, Multiple Disk Directories, Xact Copy, Epson Printer Package, Modify a COMAL 2.0 Cartridge, User Group Listing, Expert System, Keywords Printer, Sets in COMAL, Significant Digits, Multi-Label, Code Doctor Package, Gravity Simulator, 3D Projections, Extended Directory, Vocabulary, Castle Adventure, Celestial Navigation, Home Finance Calculator, IBM Packages Summary, Simple Term, Music Master Translator, Noon Sight, Right Turn Only, several Sorts, Sky View, Stars, Print the Variable Table, 1520 Calendar, Solar Eclipse Simulation, Long Function Key Definitions, Maps, Dice Routines, and Changing COMAL.

And this doesn't include several disks full of programs nearly ready for release as User Group Disks (watch next issue) or several new books being edited, each including a matching disk of programs. In addition to the articles and books, we also "edit" the programs, trying to improve them before publication. And nearly all this work is done by only two people. Perhaps if we sold more disks and books we could afford another person to help us (and we sure could use the help). If you have any suggestions or ideas, **please let us know.** As it is, we can't print very many Questions and Answers, Letters, or Tips and Notes. We miss them!

We reduced the disk subscription price. Each disk includes quality programs on both disk sides. We use premium quality, couble sided, double density disks, and mail them First Class in special boxes for the best protection during shipping. We hope many more of you will take advantage of our disk subscription. Our rates are the lowest we know of. See page 78 for the directory of last issues disk (the front side was the complete Graphics Editing System). Virtually all programs listed in the newsletter (plus some bonus ones) are on the *Today Disk.*

COMALites Unite!

by David Stidolph

Last issue we published a COMAL 2.0 program that made use of the C128 80-column video chip, and hinted that a package would soon be on the way. Well, the package has been delayed. Several people reported that their monitors did not show the 80 column screen very well (the first character was missing on some). Next issue we will publish a COMAL 2.0 program that allows you to test the new Video Display Controller with your monitor and change the register values of the VDC at will. This 80 column chip is one of the best features of the C128, and I would like to be able to use all its abilities including 640x200 pixel graphics.

We now have an EPROM chip that plugs into the empty socket inside the black COMAL 2.0 cartridge. It sets the default colors and prints a short greeting when the computer is turned on. Then if the *SHIFT* key is not pressed, it RUNs a program called *"autoboot"* from disk. A special price for the chip is $10. You must specify whose name should be displayed on power up and what 3 colors to set for default on the text screen.

This issue presents the latest extension to COMAL 0.14: *The Enhancer*. It adds extra editing abilities similar to those in COMAL 2.0, but restricts the number of sprite images to 12. While this is sufficient for *nearly* all programs, if a program attempts to use any other sprite image, the system will crash. To prevent this, add the following line at the start of such programs (stops the program without harming the system):

```
0001 if peek(806)+peek(807)*256=49920 then stop
```

What we really need are enhancements to COMAL 0.14 that are *"invisible"*, such as our expanded memory which did not restrict sprite use. Could some intrepid programmer put the error messages in memory, but place them in the hidden RAM underneath the I/O block at $d000? We also could use a list of the memory areas that are free in the $b000 block, where COMAL calls certain BASIC routines (for machine code routines).

Congratulations to the Holland COMAL Users Group. They turn out some of the most powerful programs written in COMAL, and provide them free for all to use (see *Rabbit* and *The Enhancer* this issue). When we receive a disk from them, it takes several days just to look through the material on it.

COMAL 2.0 users may be interested in *Packages Library*, a 2.0 book/ disk set. For those who **do not** want to write packages themselves, 17 packages are included, ready to use. For those who **do** write packages, source code for 13 of the packages is on the disk as well as a smooth scrolling editor for writing source code. This editor is one of the best I've ever seen, and is perfect for writing assembly code programs. Also on the disk is a program to **delink** packages (writes a LINKable file to disk), a label generating disassembler, and a file to fix some of the bugs in Commodore's Assembler. The book also explains opening files from machine code, writing macro's, and conditional assembly. The packages on the disk include: basic, bitmap, calchex, char, cmon'casbuf, cmon'rs232, compactor, demo, exeq, finchutilit, first'last, icon, meta, meta'rommed, ml, oki92, printer, and text.

0.14 - The Enhancer

by Dick Klingens & Marcel Bokhorst
Dutch COMAL Users Group

[Editors note: COMAL 0.14 is a fine language, and has remained unchanged since November 1983 when it was released. In December 1984, programming space was expanded to nearly 12K and error messages were automatically loaded into memory. All our disks from then on have included the expansion. Now, we have The Enhancer from the Dutch COMAL Group. The COMAL 0.14 system file on disk remains unchanged. The boot program adds the enhancements after it is loaded. While it has many improvements over our expanded system, it does not completely replace it since it also has added restrictions.]

The boot program **boot c64 comal** has been changed to load in our enhancement system, called *The Enhancer*. Once it is loaded, *The Enhancer* loads COMAL into memory, and then makes its changes (the COMAL 0.14 system file itself is thus unchanged).

Two versions of *The Enhancer* are included on *TODAY Disk #12*. One is for use with a 1541 drive (uses a modified version of **ml.sizzle**). The other for any other type of drive (not fastloaded). The boot program automatically senses what type of drive you are using and loads in the appropriate one.

Once COMAL is in memory the file **comalerrors** is loaded, and COMAL is changed to work with *The Enhancer*. When all updating is finished, the COMAL 0.14 program **hi** is CHAINed normally (allowing autobooting of other COMAL programs).

The Enhancer is primarily for editing programs, and adds these features (those preceded by a * are just like the 2.0 Cartridge):

+ Error messages in memory (the file **comalerrors** is used).
* A short beep when an error occurs.
* Quote mode and insert mode are disabled at the start.
+ CHR$(27)+CHR$(x) prints the error message with number x.
* CHR$(7) produces a beep.
+ CTRL-A removes indention in a line which extends over more then one screen line.
+ CTRL-K deletes all characters from the cursor position to the end of the line.
+ CTRL-U moves the cursor up to the start of the line.
* CTRL-L moves the cursor to the last character on the line.
* CTRL-V sets up colors on the textscreen: border 6, background 6, penclor 1.
* CTRL-W sets up border 11, background 15, pencolor 0 on the textscreen.

The function keys are defined to type text when pressed. Up to 10 characters can be defined for each function key (the limit of the keyboard buffer). The SHIFT/ RUN key is also definable (as function key number 0). The default settings of the function keys are similar to those with the 2.0 Cartridge:

```
 f1 - (unchanged)
 f2 - PASS "i0"
 f3 - (unchanged)
 f4 - AUTO
 f5 - (unchanged)
 f6 - LIST
 f7 - RUN + chr$(13)
 f8 - RENUM + chr$(13)
SHIFT+RUN/STOP - CHAIN "*" + chr$(13)
```

More ►

Because *The Enhancer* occupies much of the memory in the $c000 area, only sprite definitions 0-11 can be used. **Warning:** If you attempt to define a sprite shape higher than 11, the system will crash. You can still edit programs that use those higher numbered sprite images - just don't run the program. The program can be saved and then run later from the normal COMAL 0.14 system. On boot up, you will be asked if you want *The Enhancer* included. If you reply no, our expanded system will be loaded as usual.

The routine for loading the error messages is placed from 51500 ($c95e), so with SYS 51500 it is possible to reload the messages, unless the code is overwritten by sprite information.

The function key definitions are stored starting at $c544 in 9 parts of 10 characters. The are stored in the order of their CHR$ values (1,3,5,7,2,4,6,8). The SHIFT- RUN/ STOP key is also definable (as function key number 0).

We developed some COMAL 0.14 procedures to allow you to produce sounds, adapt your function keys, etc. These include **defkey, showkeys, bell,** and **quote'mode.** Each of these procedures are on *Today Disk #12* in LISTed format, so you can ENTER them into whatever programs you write.

Finally, I would like to mention the name of the programmer, Marcel Bokhorst, because he coded *The Enhancer* all by hand using a monitor program (he lacked a good assembler). ■

Rabbit 2.0

by Marcel Bokhorst

The package RABBIT (on *Today Disk #12*), has three features:

1. Speeds up COMAL program loading
2. Speeds up COMAL file reads
3. Fast disk block read & write

Program loads, EXTERNAL procedure calls, and disk file access times are cut in half. Direct sector read and write times are cut by 75%.

The package is located at $7000-$7ccb, so FONTs cannot be used (the system will crash if you try). The package will automatically initialize itself when it is first LINKed.

When active, reading or writing to disk in fast mode will cause the screen to flash on and off. This is normal. Certain operations, like CAT, will NOT cause the screen to flicker because FASTMODE is not used.

RESTRICTIONS

This package will only work on the 1541. Do not try it with any other type of disk drive (such as MSD). If the package "crashes", you will have to turn your computer off and restart it.

FONTs may not be loaded or linked while this package is in memory. If this is tried, the computer will crash and the power will have to be switched off and back on. Other commands in the FONT package (such as GETCHARACTER) will still work.

Program space is reduced to 26,618 bytes, and large programs may not fit into the reduced memory area.

More ►

COMMANDS

USE rabbit - Initializes the package and adds the new commands listed below to the name table.

BREAD(track,sector,buffer,string$) - Block read from the track and sector specified using the specified buffer into string$ (a 256 byte string).

BWRITE(track,sector,buffer,string$) - Block write to the track and sector specified using the specified buffer with string$ (a 256 byte string).

SETFAST(true/false) - Sets if you want to do FAST reads or writes. TRUE indicates you want FASTMODE on, and FALSE indicates you want normal speed, and also cancels BMODE (below).

FAST - Returns TRUE if FASTMODE is enabled, FALSE if disabled.

SETBMODE(true/false) - Sets whether you want to do BLOCK reads and writes using the FAST mode.

BLOCKMODE - Returns TRUE if blockmode is enabled, FALSE if disabled.

SETROMMED(true/false) - If set to true a discard is ignored and the package will remain in the computer (and will not be saved along with programs). When set to false it will be saved along with programs and both NEW and DISCARD will delete it.

ROMMED - Returns TRUE if package is ROMMED, FALSE if not.

SETDEVICE(num) - Sets the drive number to be used (8,9,10,11, etc.).

DEVICE - Returns the current device number.

TRACK - Returns the number of the next track (based on the last sector read with bread).

SECTOR - Returns the number of the next sector (based on the last sector read with bread).

RDERR - Returns the error number for last read/write operation (a disk error will NOT generate a standard error for TRAP .. HANDLER).

EXAMPLE OF USE

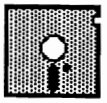
The following simple program shows how easy it is to double the speed of reading data from disk in your programs. All your file access statements remain unchanged. You only need to add the USE rabbit at the start of the program and turn on fast mode (setfast(TRUE)).

```
USE rabbit
setfast(TRUE) //optional- true is default
DIM text$ OF 80
PAGE
PRINT "Your current device is:";device
INPUT "Filename to type out: ": name$
OPEN FILE 2,name$,read
WHILE NOT EOF(2) DO
  INPUT FILE 2: text$
  PRINT text$
ENDWHILE
END "All Done." ■
```

THE ↑ STRIKES AGAIN

There was a typographical error in the article, *Batch Files From Memory* on page 32 of *COMAL Today #7*. A copyright symbol was printed instead of the ^ character in the INPUT line inside the SETUP procedure. ■

Font Editor 0.14

by Phyrne Bacon 

The 0.14 font editor is a simplified version of the 2.0 font editor. It can load either "font." or "set." fonts, and can save a font as a "set." font. It allows you to choose the character to edit by key, cursor or screen code. The font editor displays all 256 characters of the font. The program is on *Today Disk #12.* **Note:** Do not use this program with *The Enhancer.*

GETTING STARTED

This font editor requires the expanded memory option for COMAL 0.14. After COMAL has been loaded, type new, and then type size. If it says, "11838 bytes free", you have memory expanded and may load FONT EDITOR. If not, type chain"hi". After HI runs, return to COMAL, type new and then size. If all else fails, turn the computer off briefly, and try again. HI is run automatically if the boot is used to load COMAL. All the more recent versions of HI expand the memory.

LOADING THE FIRST FONT

After you type RUN, you are asked if you want to edit a "font." or a "set." font. (Each "font." file has two fonts: and alphafont and a betafont. Usually an alphafont is an uppercase font or a custom font, and a betafont is upper/lowercase.) If you pick "font.", you will be asked if you want to edit the alphafont or the betafont. To change to another font, see LOADING A FONT below.

There are forty-one "font." files and thirty-six "set." files on the COMAL Font Disk (*see order form on page 80*).

EDIT MODE

After the font is loaded, the computer will change from a screen with the built-in font to a new screen with the loaded font. The new screen will have a large representation of the letter **a** in the upper left hand corner in the design frame (a small letter a appears to the left of the design frame, and the screen code of the letter **a** appears to the right of the design frame), a partial list of the font editor commands in the upper right hand corner (press **i** to see the complete list), and a display of all 256 characters in the cursor select frame in the lower half of the screen.

If the cursor is hard to see, you can change 3 (cyan) on line 1400 to 15 (light gray), 13 (light green), or some other color, or you can move the cursor a little with the cursor keys: a moving cursor is easier to see.

EDITING A CHARACTER

When you begin, the cursor will be in the upper left hand corner of the design frame. To change a pixel, move the cursor to the pixel and press **3** to turn the pixel on, or **4** to turn the pixel off. Use **d** to delete the row the cursor is on, or **e** to erase the column the cursor is on. The design frame has wrap: if the cursor goes off the right edge, appears in the left column, if it goes off the bottom edge it appears in the top row, and so on.

When a pixel is changed, it will be changed in the design frame, in memory, and in every copy of the character on the screen.

More ►

To shift a character upward, use **+**, to shift it down, use **-**, to shift it left use *****, to shift it right, use **/**. These shifts (rolls) are nondestructive: what goes over the top edge, appears on the bottom row, and so on.

CHOOSING A CHARACTER

To move to the following character, press **f5**. To move to the preceding character, press **f7**. To select a (nonreverse) character by pressing its key, first press **f1**. To use the cursor to select a character, first press **f3**. The cursor will move to the cursor select frame (the font display). Move the cursor to the desired character and press return. (The cursor select frame has wrap.) To select a character by screen code, first press **f2**.

COPYING A CHARACTER

To copy a character, press **c** (for copy). You will be asked to move the cursor to the character to be copied, and to press return, and then to move the cursor to the character to be copied to, and to press return. The new copy will then appear in the design frame.

REVERSE CHARACTERS

On most fonts, characters 128-255 are the reverses of characters 0-127. For example, reverse-**a** (129) has a pixel off wherever a (number 1) has a pixel on, and vice-versa. The reverse characters are used in the blinking cursor.

To make a reverse of a newly designed character, press **r**. This copies the reverse of the current character n onto n+128 (onto n-128 if n>=128), and moves you to the new reverse character.

To save time, you can change any of the characters 0-127, and then make reverse copies of all of the characters 0-127 at once by pressing **f4**.

Since reverse period is used in the character design frame, if there are no reverse characters in a font, press **f4** to make the character design suddenly appear in the design frame.

EXITING FONT EDITOR

To exit FONT EDITOR, press **z** (which ends the program and returns you to the usual built-in font), or the STOP-key (which leaves you in the custom font). To return to both the program and the font after any exit or error message, type run, or, if you used the STOP-key, you can type con.

SAVING A FONT
(usual load address)

To save a font with the usual load address, press **s**. You will be asked the name ("set." will have been typed in for you - do not type the "set."). When asked about the load address, press **u** or carriage return. The font will be saved as a 9 block PRG-file. If a file with that name already exists, the program will end. (This doesn't harm the font.) Then, if desired, you can then type cat, scratch the existing file, type run, and save the font with that name, or you can just type run, press **s** and save the font with another name.

It is best to save temporary versions of a font frequently to avoid loss of designs due to power failures and other difficulties.

More ►

WORKING ON THE DISK DIRECTORY

You can exit the program at any time by pressing **z** or the STOP-key, type cat, scratch or rename any files, read any disk directories and so on, and return to the program by typing run (or con, if you used the STOP-key). Stopping the program this way does not harm the font.

LOADING A FONT

To load a "set." font press **l** (lower case L). To load a "font." font, press **k**. You will be asked for the name (the "set." or "font." will have been already typed in for you). If you are loading a "font." font, you will be asked if you want the alphafont or the betafont. Before the font is loaded, the new fontname will be displayed.

GETTING GARBAGE ON THE SCREEN

If you press the Commodore key and the shift key at the same time while working on a font, you will get garbage on the screen. This is because the computer is using the non-existent second custom font for its character designs. To return to the readable custom font, press the Commodore key and the shift key at the same time. These keys toggle back and forth between the custom font and the non-existent second custom font.

SAVING A FONT
(with custom load address)

You only need a custom load address if you are planning to use the font with a BASIC program. To use a custom font (and the custom font textscreen) with a COMAL 0.14 program, use the program LOAD FONT as a starter, and then add whatever other programming you wish.

Press **q** to find out the load address of any PRG-file. (This can be fun.) The load address will be given as a multiple of 256; for example, 8*256+0.

To save a font with a custom load address, press **s** (for save), enter the filename, and press **c** (for custom load address). You will be asked for the load address as a multiple of 256. If the load address is 8*256, enter 8. Any real load address will be a multiple of 8*256: for example, 8*256, 16*256, 32*256 and so on. If the load address is divisible by 16*256, the font is being used as an alphafont; otherwise, it is being used as a betafont.

Sideways

by D. Shellenberg

Here are two programs to print text files (such as a Multiplan spreadsheet) sideways on a Gemini 10x or compatible printer. Both are on *Today Disk #12* and one is listed below.

Sideways60 can print up to 60 lines across a page using printer graphics with a very nice font, but is slow (12.5 minutes for 60 255 character lines).

Sideways80 uses a character set downloaded to the printer. It can print 80 lines across a page twice as fast as *Sideways60*, but not as pretty of a font.

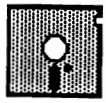
```
// delete "sideways60"
// save  "sideways60"
//  by D. Shellenberg
PAGE
PRINT AT 4,7: "A SIDEWAYS PRINTING PROGRAM"
PRINT AT 6,8: "For MULTIPLAN Print Files"
PRINT "or any text file of 60 lines or less"
PRINT "with max line length of 255 characters"
PRINT "using a C-64 and Gemini 10X printer."
PRINT "Set printer interface to NO translation"
PRINT
get'filename
get'bottomline
build'ascii'string
load'array
DIM line$(bottomline#) OF 255
PAGE
PRINT "Press ESC key to quit during printing..."
OPEN FILE 3,"lp:",WRITE
TRAP ESC-
// change linefeed to 8/72 inch
PRINT FILE 3: ""27"a"8""
WHILE NOT EOF(2) DO
 in'ln#:=0
 longest#:=1
 WHILE NOT EOF(2) AND in'ln#<bottomline# DO
  in'ln#:+1
  INPUT FILE 2: line$(in'ln#)
  IF LEN(line$(in'ln#))>longest# THEN
   longest#:=LEN(line$(in'ln#))
  ENDIF
 ENDWHILE
 fin'in'ln#:=in'ln#
 gfx'ctr'hi#:=in'ln#*8 MOD 256
 gfx'ctr'lo#:=INT(in'ln#*8/256)
 FOR pt'ln#:=1 TO longest# DO
  PRINT FILE 3: ""27"k",
  PRINT FILE 3: CHR$(gfx'ctr'hi#),
  PRINT FILE 3: CHR$(gfx'ctr'lo#),
  FOR pt'chr#:=fin'in'ln# TO 1 STEP -1 DO
   check'esc
   IF pt'ln#<LEN(line$(pt'chr#))+1 THEN
    print'one'char
   ELSE
    FOR x:=1 TO 8 DO
     PRINT FILE 3: ""0"",
    ENDFOR x
   ENDIF
  ENDFOR pt'chr#
  PRINT FILE 3: "" // new line
 ENDFOR pt'ln#
ENDWHILE
// change linefeed back to normal
PRINT FILE 3: ""27"a"12""
CLOSE
END "Finished"
//
PROC build'ascii'string
 DIM ascii'str$ OF 123
 ascii'str$:=""
 FOR n#:=0 TO 12 DO
  ascii'str$:+CHR$(0)
 ENDFOR n#
 ascii'str$:+CHR$(13)
 FOR n#:=14 TO 31 DO
  ascii'str$:+CHR$(0)
 ENDFOR n#
 ascii'str$:+" !""#$%&'()*+,-./"
 ascii'str$:+"0123456789:;<=>?@"
 ascii'str$:+"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
 ascii'str$:+"[\]^'\abcdefghij"
 ascii'str$:+"klmnopqrstuvwxyz"
ENDPROC build'ascii'string
//
PROC print'one'char
 p:=line$(pt'chr#)(pt'ln#) IN ascii'str$
 PRINT FILE 3: table$(p-1),
ENDPROC print'one'char
//
PROC get'filename
 LOOP
  PRINT AT 13,1: "File name (RETURN to cancel)"
  INPUT "": filename$,
  IF filename$="" THEN END "You pressed RETURN to cancel"
  TRAP
   OPEN FILE 2,filename$,READ
   PRINT AT 24,1: SPC$(39), //clear error msg
   EXIT
  HANDLER
   IF ERR=262 THEN // file not found
    PRINT AT 24,1: "Can't find";
    PRINT filename$,", try again",
   ELSE
    PRINT AT 24,1: ERRTEXT$,
   ENDIF
  ENDTRAP
 ENDLOOP
ENDPROC get'filename
//
```

More ►

Sideways - continued

```
PROC get'bottomline
  LOOP
    PRINT AT 16,1: "Bottom line number (RETURN to cancel)"
    INPUT "": bottomline#
    IF bottomline#>60 THEN
      PRINT "Sorry, the file cannot be longer than 60 lines."
    ELSE
      EXIT
    ENDIF
  ENDLOOP
ENDPROC get'bottomline
//
PROC check'esc
  IF ESC THEN
    // change linefeed back to normal
    PRINT FILE 3: CHR$(27)+CHR$(64)
    CLOSE
    TRAP ESC+
    END "You pressed escape!!!"
  ENDIF
ENDPROC check'esc
//
PROC load'array
  DIM table$(32:126) OF 8
  FOR row#:=32 TO 126 DO
    FOR col#:=1 TO 8 DO
      READ num#
      table$(row#)(col#):=CHR$(num#)
    ENDFOR col#
  ENDFOR row#
  DATA 0,0,0,0,0,0,0,0
  DATA 0,8,0,0,8,8,8,8
  DATA 0,0,0,0,0,20,20,20
  DATA 0,36,36,126,36,126,36,36
  DATA 0,8,62,9,62,72,62,8
  DATA 0,71,37,23,8,116,82,113
  DATA 0,115,72,84,34,80,80,96
  DATA 0,0,0,0,0,16,8,24
  DATA 0,2,4,8,8,8,4,2
  DATA 0,32,16,8,8,8,16,32
  DATA 0,66,36,24,255,24,36,66
  DATA 0,0,8,8,62,8,8,0
  DATA 16,8,24,0,0,0,0,0
  DATA 0,0,0,0,62,0,0,0
  DATA 0,24,24,0,0,0,0,0
  DATA 0,64,32,16,8,4,2,1
  DATA 0,92,34,81,73,69,34,29
  DATA 0,28,8,8,8,8,24,8
  DATA 0,63,32,16,14,1,33,30
  DATA 0,30,33,1,2,4,2,31
  DATA 0,2,2,63,34,18,10,6
  DATA 0,30,33,1,1,62,32,63
  DATA 0,62,65,65,62,32,16,8
  DATA 0,32,16,8,4,2,1,63
  DATA 0,62,65,65,62,65,65,62
  DATA 0,16,8,4,62,65,65,62
  DATA 0,48,48,0,0,48,48,0
  DATA 32,16,48,48,0,48,48,0
  DATA 0,4,8,16,32,16,8,4
  DATA 0,0,0,126,0,126,0,0
  DATA 0,32,16,8,4,8,16,32
  DATA 0,16,0,16,12,2,66,60
  DATA 0,31,32,39,41,39,33,30
  DATA 0,129,129,255,129,66,36,24
  DATA 0,252,66,66,124,66,66,252
  DATA 0,60,66,64,64,64,66,60
  DATA 0,248,68,66,66,66,68,248
  DATA 0,127,64,64,126,64,64,127
  DATA 0,64,64,64,126,64,64,127
  DATA 0,62,65,65,71,64,64,62
  DATA 0,65,65,65,127,65,65,65
  DATA 0,28,8,8,8,8,8,28
  DATA 0,24,36,4,4,4,4,30
  DATA 0,66,68,72,112,72,68,66
  DATA 0,127,64,64,64,64,64,64
  DATA 0,65,65,73,73,93,99,65
  DATA 0,65,67,69,73,81,97,65
  DATA 0,62,65,65,65,65,65,62
  DATA 0,64,64,64,126,65,65,126
  DATA 0,61,66,69,65,65,65,62
  DATA 0,65,66,68,124,66,66,124
  DATA 0,62,65,1,62,64,65,62
  DATA 0,8,8,8,8,8,8,127
  DATA 0,62,65,65,65,65,65,65
  DATA 0,8,28,20,34,65,65,65
  DATA 0,34,54,93,73,65,65,65
  DATA 0,65,34,20,8,20,34,65
  DATA 0,8,8,8,8,20,34,65
  DATA 0,127,32,16,8,4,2,127
  DATA 0,28,16,16,16,16,16,28
  DATA 0,1,2,4,8,16,32,64
  DATA 0,56,8,8,8,8,8,56
  DATA 0,0,0,0,0,34,20,8
  DATA 0,127,0,0,0,0,0,0
  DATA 0,0,0,0,0,8,16,24
  DATA 0,125,66,62,2,60,0,0
  DATA 0,124,66,66,66,124,64,64
  DATA 0,60,64,64,64,60,0,0
  DATA 0,30,34,66,34,30,2,2
  DATA 0,60,64,126,66,60,0,0
  DATA 0,16,16,16,60,16,16,14
  DATA 30,1,63,65,65,62,0,0
  DATA 0,34,34,34,50,46,32,32
  DATA 0,28,8,8,8,24,0,8
  DATA 112,8,8,8,8,24,0,8
  DATA 0,66,116,72,68,66,64,64
  DATA 0,28,8,8,8,8,8,24
  DATA 0,73,73,73,73,126,0,0
  DATA 0,34,34,34,50,46,0,0
  DATA 0,60,66,66,66,60,0,0
  DATA 64,64,124,66,66,124,0,0
  DATA 2,2,62,66,66,62,0,0
  DATA 0,32,32,32,50,44,0,0
  DATA 0,62,1,62,64,62,0,0
  DATA 0,12,16,16,16,60,16,16
  DATA 0,58,68,68,68,68,0,0
  DATA 0,24,24,36,66,66,0,0
  DATA 0,34,99,93,73,65,0,0
  DATA 0,34,20,8,20,34,0,0
  DATA 32,16,8,20,34,34,0,0
  DATA 0,60,32,16,8,60,0,0
  DATA 0,12,16,16,32,16,16,12
  DATA 0,8,8,8,0,8,8,8
  DATA 0,24,4,4,2,4,4,24
  DATA 0,0,0,0,0,6,73,48
ENDPROC load'array
```

Double Column File Printer

by Jim Ventola 

Although it is possible to print in double columns using Easy Script, it requires manually re-positioning the paper for the second column. Since I have never been able to get the paper in exactly the right place, the second column never quite lined up with the first. So I was excited when I received *User Group Disk #9* (COD newsletter on disk) and discovered that its MENU program (by Colin Thompson and others) was able to read files made with Easy Script and print them in double columns directly. The problem was that the MENU program is a powerful SYSTEM for publishing an on-going newletter and was not designed simply to print in double columns. So it reads in arrays of articles' names, does fancy stuff with screen colors, requires a directory file of the articles to show or print, etc. All I wanted was to print my files in double columns. So I became a "hacker," chopping away at the code and making some changes along the way.

It is surprisingly difficult to cut down a program written far above your own programming abilities. But COMAL makes it feasible because the programming environment is so powerful and easy to use and the code is so readable (and easy to steal). I took the plunge; *Doubler* is the result.

Here's how Doubler works. Make your files with your wordprocessor (SEQ type text files). From Doubler, they can be viewed on screen or printed out in double columns of 32 characters. There are four "fields" at the top of the page, then your text (up to 50 lines per page), and then one last "field" at the foot of the page. You can have a field be blank, contain a ruler made up of some repeating character, contain text, and/or have automatic pagination.

The file *"doubler.doc"* contains full instructions and is itself an example of the sort of file Doubler prints out. It is on *Today Disk #12* along with *Doubler*.

```
// (c) jan 1986 by James Ventola
// All rights reserved.
// Permission granted to give
// away but not sell this
// program if the documentation
// file accompanies it.
dims
initialize'system
main'menu
//
proc move'bar(ref array$(),max'row)
 cursor(22,8)
 print "Use CRSR and <RETURN>"
 selected:=false
 move'it:=false
 case direction of
 when -1
  if row>4 then
   new'row:=row-1
   move'it:=true
  endif
 when +1
  if row<max'row then
   new'row:=row+1
   move'it:=true
  else
   new'row:=4
   move'it:=true
  endif
 when 2
  selected:=true
 otherwise
 endcase
 if move'it then
  cursor(row,3)
  print chr$(146);" ",array$(row-3),tab(28),
  cursor(new'row,3)
  print chr$(18);" ",array$(new'row-3),tab(28),
  row:=new'row
 else
  cursor(row,3)
  print chr$(18);" ",array$(row-3),tab(28)
 endif
endproc move'bar
//
func direction
 option$:=key$
 case option$ of
 when chr$(17)
```

More ►

```
 return +1
 when chr$(145)
  return -1
 when chr$(13)
  return 2
 otherwise
  return 0
 endcase
endfunc direction
//
proc initialize'system
 background 6
 border 14
 pencolor 1
 print chr$(14) // lowercase mode
 close
 select output "ds:"
 stat$:=status$
 row:=4
 new'row:=4
 selected:=false
 array$(3):="Directory of SEQ files"
 array$(1):="Show file on screen"
 array$(2):="Print it out"
 array$(4):="General instructions"
 array$(5):="Back to COMAL"
 prompt$:="        << Hit any key  >>"
 men'max:=5
endproc initialize'system
//
proc dims
 dim prompt$ of 30
 dim pagination(5)
 max:=5
 dim char$ of 1
 dim head'title$(5) of 69
 dim name$ of 16
 dim array$(9) of 30
 dim text$ of 39
 max'lines:=50
 dim left'column$(max'lines) of 39
 dim option$ of 1
 dim s$ of 24, stat$ of 30
 max'row:=4
endproc dims
//
proc main'menu
 page
 cursor(5,15)
 print "ASCII File"
 cursor(6,10)
 print "Two Column Printer*"
 cursor(7,13)
 print "by Jim Ventola"
 cursor(23,1)
 print prompt$
 cursor(15,5)
 print "*from Colin Thompson's COD MENU"
 wait
 repeat
  display'options(array$,men'max+3)
  repeat
   move'bar(array$,men'max+3)
  until selected
  execute'option(row)
 until false
endproc main'menu
//
proc display'options(ref array$(),max'options)
 page
 cursor(4,1)
 for x:=1 to max'options do
  print tab(5),array$(x)
 endfor x
 row:=4
endproc display'options
//
proc execute'option(row)
 trap esc-
 case row-3 of
 when 4
  file'to'screen("doubler.doc")
 when 3
  get'directory("0:*=seq",8)
 when 1
  screenview
 when 2
  print'it
 when 5
  trap esc+
  exit'to'comal
 otherwise
 endcase
 while esc do null
 trap esc+
endproc execute'option
//
proc screenview
 page
 print tab(8),chr$(18),"V I E W  O N  S C R E E N"
 cursor(6,1)
 print "To abort, hit RETURN"
 cursor(10,1)
 input "Filename: ": name$
 if len(name$) then
  file'to'screen(name$)
 endif
endproc screenview
//
proc exit'to'comal
 close
 page
 cursor(10,9)
 print "May you go in joy..."
 cursor(22,1)
 end
endproc exit'to'comal
//
proc error
 page
 box
 print chr$(17),chr$(29),"Error:";chr$(18),s$
 cursor(12,3)
 print " Check the filename again"
 cursor(22,11)
endproc error
```

More ►

```
//
proc cursor(row,col) closed
 poke 211,col-1
 poke 209,(1024+(row-1)*40) mod 256
 poke 210,(1024+(row-1)*40) div 256
 poke 214,row-1
endproc cursor
//
proc shift'wait
 shift'flag:=653; color:=13
 while not peek(shift'flag) and not eof' do
  color:=(color+8) mod 16
  pencolor color
  print "Press SHIFT to continue, or ^ to quit",chr$(145)
  if key$="^" or esc then eof':=true
 endwhile
 pencolor (1)
 string(" ",37)
 print chr$(145)
endproc shift'wait
//
proc file'to'print(name$)
 if file'exists(name$) then
  open file 5,name$,unit 8,read
  open file 255,"",unit 4,7,write
  select output "lp:"
  page'num:=0
  repeat
   for i:=1 to 40 do left'column$(i):=""
   eof':=false
   full:=false
   read'left'column
   print'a'page
  until eof'
  close file 5
  select output "ds:"
  pass "i0:"
 else
  error
 endif
endproc file'to'print
//
proc check'for'eof
 if esc then eof':=true
 if key$="^" then eof':=true
 if line'count=max'lines then full:=true
endproc check'for'eof
//
proc read'left'column
 page'num:+1; line'count:=0
 repeat
  if not eof(5) then
   line'count:+1
   input file 5: left'column$(line'count)
   k:=len(left'column$(line'count))
   if k then
    left'column$(line'count):=left'column$(line'count)(3:k)
   endif
   check'for'eof
  else
   eof':=true
  endif
 until eof' or full
endproc read'left'column
//
proc print'a'page
 page'header
 if eof' then
  for i:=1 to line'count do print tab(8),left'column$(i)
 else
  print'line:=0
  repeat
   if not eof(5) then
    print'line:+1
    input file 5: text$
    k:=len(text$)
    if k then text$:=text$(3:len(text$))
    print tab(8),left'column$(print'line),tab(44),text$
    check'for'eof
   else
    eof':=true
    for i:=print'line+1 to max'lines do
     print tab(8),left'column$(i)
    endfor i
   endif
  until print'line=max'lines or eof'
  rt'col'count:=print'line
 endif
 page'footer
endproc print'a'page
//
proc wait
 while key$<>chr$(0) do null
 while key$=chr$(0) do null
endproc wait
//
proc page
 print chr$(147),
endproc page
//
proc line'up'paper
 page; cursor(12,4)
 print "Align the paper near the top of form"
 cursor(14,6)
 print "Press any key to begin printing"
 wait; page
 cursor(12,4)
 print "Press the ^ key to stop printing"
 print "(Printing stops at the end of the page)"
endproc line'up'paper
//
proc file'to'screen(name$)
 page
 print "Going to disk..."
 if file'exists(name$) then
  open file 5,name$,unit 8,read
  print chr$(147),chr$(14)
  cursor(25,1)
  eof':=false
  repeat
   if not eof(5) then
    input file 5: text$
    print text$
    shift'wait
   else
    eof':=true
```

More ►

Double Column File Printer - continued

```
   endif
  until eof'
 else
  error
 endif
 close'files
endproc file'to'screen
//
proc close'files
 close file 5
 select output "ds:"
 print prompt$
 wait; page
endproc close'files
//
proc box
 print chr$(19), chr$(176),
 string(chr$(192),37)
 print chr$(174)
 print chr$(221),tab(39),chr$(221)
 print chr$(173),
 string(chr$(192),37)
 print chr$(189),chr$(19),
endproc box
//
proc open'file(filename$)
 if file'exists(filename$) then
  open file 78,filename$,read
 else
  error
 endif
endproc open'file
//
func file'exists(prog$)
 open file 78,prog$,read
 s$:=status$
 close file 78
 return s$(1:2)="00"
endfunc file'exists
//
proc get'directory(name$,device)
 // by David Stidolph
 page
 for x:=1 to len(name$) do
  poke 834+x,ord(name$(x))
 endfor x
 poke 26997,x
 poke 27013,device
 poke 27110,96
 sys 26996
 poke 27110,76
 poke 26997,1
 poke 27013,8
 print prompt$
 wait
endproc get'directory
//
proc print'it
 page
 for v:=1 to 5 do pagination(v):=0
 for t:=1 to 5 do head'title$(t):=""
 cursor(5,1)
 print "There are four fields at the top"
print "and one at the foot."
print "For a blank line, select"
print "either a ruler or a headline"
print "and enter a space for its character."
print
print
print "To abort, hit RETURN at filename prompt."
for i:=1 to 5 do
 print "Is field ";i;" a ruler or a headline? "
 input "r/h ": option$
 case option$ of
 when "r"
  input "Ruler of what character?  ": char$
  for t:=0 to 68 do head'title$(i)(t):=char$
  print head'title$(i)
 when "h"
  print "Enter the text:"
  input head'title$(i)
  print head'title$(i)
  print "Does this field print the pagenumber?"
  input "It takes 10 spaces. n/y ": option$
  if option$="y" then
   pagination(i):=true
  endif
 otherwise
  null
 endcase
endfor i
input "What's the filename? ": name$
if len(name$)>0 then
 line'up'paper
 file'to'print(name$)
endif
endproc print'it
//
proc page'header
 print
 for i:=1 to 4 do
  if pagination(i)=true then
   print tab(8),head'title$(i);
   print tab(67);"Page:";page'num
  else
   print tab(8),head'title$(i)
  endif
 endfor i
 for t:=1 to 2 do print
endproc page'header
//
proc page'footer
 for t:=1 to (53-line'count) do print
 if pagination(5)=true then
  print tab(8),head'title$(5);
  print tab(67);"Page: ",page'num
 else
  print tab(8),head'title$(5)
 endif
 for times:=1 to 5 do print
endproc page'footer
//
proc string(char$,num)
 for temp:=1 to num do print char$,
endproc string
```

FOURIER TRANSFORMATION

by Lowell Zabel 

In many engineering studies the amplitudes of the frequency components of periodic variables are desired. This is one of the most straightforward applications of the Fourier integral. My program (on *Today Disk #12*) assumes that the data is in the form of equally spaced points on the input curve. A detailed treatise on the Fourier integral is well beyond the scope of this introduction. Therefore, I will limit this discussion to the programs limitations, strengths and use.

DATA ENTRY

The values at each sampling point may be entered manually during the program run by entering each value as requested. This mode is selected from the menu. Following entry you will be asked if the data is to be saved on the disk and if so to supply a file name. I strongly recommend that you save each set of data as entry can be tedious. Any file can be overwritten with new data at any time. The other way to enter data is to copy a file already on disk. A file can be prepared separate from this program and listed to disk. The program will then read this file on demand from the menu. During the run, you will be asked for the length of the file in time units. These units may be seconds, minutes or eons, as long as you realize that the frequency scale will then be in cycles per second, cycles per minute or cycles per eon.

CALCULATIONS

The calculations use the brute force and awkwardness method using sines and cosines. The details of the method can be reviewed in any good book on numerical methods. I have used the trapazoid method of integration without end corrections. The inclusion of end corrections would not make an observable improvement in the curves produced. The frequency scale on the plot is automatically selected. I have arbitrarily set the minimum number of data points per cycle at the upper frequency limit at six. Six is about the practical lower limit for definition of a sine curve. Even here you will notice excessive scatter at the upper end of the curve for some sets of input data.

The running time is tolerable and depends on the number of data points being integrated. Twenty points will run in about 10 minutes while 500 points will require well over an hour.

Because of the time required to make the calculations and to plot the curve, automatic scaling of the amplitude axis has not been included. Instead you will be asked to estimate the amplitude of the highest peak and enter this value. The amplitude scale will be adjusted accordingly. The full scale values run from 1 to 8. If the maximum amplitude is expected to be less than .5 or more than 8 then you should multiply all of your input data values by an appropriate factor before entering them into the program.

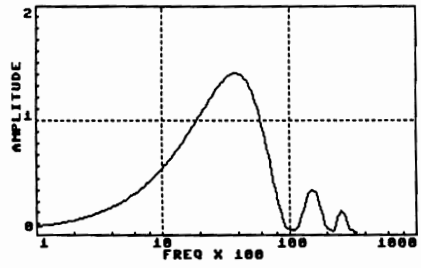
SAMPLE DATA SETS

I have included two sets of data on this disk. SQUARE.DAT is a square wave consisting of one cycle. Twenty points are used. I suggest that a maximum peak value of 1.9 be used and that a length of 2 be used for the time length of the data. You will note that the peaks are

More ►

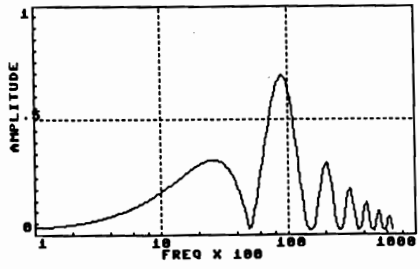
not exactly on the expected frequencies of .5, 1.5, 2.5, etc. The peaks will be at slightly lower frequencies than these because the input data includes only one cycle. An infinite number of cycles would be required to produce exact frequencies. To run these data use option 2 in the menu. The resulting curve is also on the disk and may be accessed by typing:

```
use graphics
loadscreen("hrg.square")
```



SAWTOOTH.DAT is two cycles of a sawtooth wave. The high resolution screen is accessed with:

```
use graphics
loadscreen("hrg.sawtooth")
```



If you wish to run these data use a time of 2 and a maximum value of .7 for the sawtooth curve. Both of the above curves exhibit peaks with maximum amplitudes close to the theoretical values. ■

Best Sellers

We are happy to see that *COMAL Yesterday* is still on the chart. The first four issues of *COMAL Today* contained plenty of good information, and lots of small tips and notes. Since COMAL 0.14 has not changed since it was released, all this information still applies. It looks like our readers realize that!

Now, that there finally is an *American* text book introduing COMAL, *Beginning COMAL* will have some steep competition. The new text is by West Virginia professor J William Leary. His book, *"Introduction to Computer Programming With COMAL"* is spiral bound for easy use, and has a separate answer book (teachers can choose whether or not the students have the answer book).

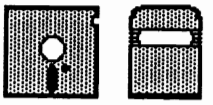
February 1986

#1 - COMAL From A To Z
by Borge Christensen
#2 - Cartridge Tutorial Binder
by Frank Bason & Leo Hojsholt
#3 - Cartridge Graphics and Sound
by Captain COMAL's Friends
#4 - COMAL Yesterday
First Four COMAL Today issues
- COMAL Handbook
by Len Lindsay
#5 - Beginning COMAL
by Borge Christensen

March 1986

#1 - COMAL From A To Z
by Borge Christensen
#2 - COMAL Workbook
by Gordon Shigley
#3 - COMAL Handbook
by Len Lindsay
#4 - COMAL Yesterday
First Four COMAL Today issues
#5 - Cartridge Tutorial Binder
by Frank Bason & Leo Hojsholt ■

A Users Group Meeting



by Ed Matthews

I introduced COMAL 0.14 at our last CUGOS meeting, showing a graphics program and an installment loan payment program. I passed out copies of the program listings, ran the programs, then showed that the procedures were still usable in direct mode. Since this was a large group with diverse interests, I didn't feel I could spend much time on the language, itself, so I quit after the fun parts. However, there are fifty or so people who know why you shouldn't buy a washer and dryer with your new house, now that they have seen $500 amortized over thirty years. I also introduced Calvin and showed FILL, HIDETURTLE, and TURTLESIZE. I may do sprites and another data manipulation next month.

After the meeting, I was astonished at how many people came to me to ask questions. I had mentioned *COMAL from A to Z* during the presentation, and several people asked me to order copies for them. The club librarian tells me we are moving quite a few of the Sampler and Tutorial disks, as well as others. Referring to the program listings I had passed out, a number of people asked, *"What disk do I have to get just so I can type in these programs?"* To accomodate them I plan to put together a "working disk," so people will have a COMAL System, fast loader, and a few examples all on one disk, also information, instructions for 0.14, and Logo book sample. If I like it, I'll send you a copy.

At our meetings, we have had a lot of demonstrations (good ones, from good people) of equipment and programs I doubt I'll ever have the "extra" money to buy. COMAL 0.14 is a product any of us can afford. I plan to introduce COMAL 0.14 a little at a time, and emphasize its simplicity, versatility, and **value**, just to get people to try programming. After they are hooked, we can talk about the more powerful COMAL, and you can start feeding your family on a more regular basis. (Do you have another job? I've been wondering about you, Jim Butterfield and Borge Christensen.) In the meantime, I expect to be hated by the developers of Logo, Doodle, Simon's, Power,... (I'm innocent! It's all Borge's fault! I just use it!)

The only thing I have found seriously wrong with *COMAL Today* (even the back issues I've read several times) is I get so excited over some ideas presented I almost hyperventilate. (Maybe it should have a warning label).

In *COMAL Today #10*, you have repeated your request that we share several specified disks, but not book disks or "*special collection disks*". I am not certain what this last group includes. You might consider putting an asterisk beside the disks that we should share freely on the order blank at the end of each *COMAL Today*, and make the message sound positive, rather than negative.

*Examples: "*Please feel free to copy and share the contents of disks marked with asterisks with your friends, schools, and clubs. We ask that you not sell these programs or the COMAL System without permission from COMAL Users Group USA, Ltd., and that you understand that the disks not marked may be copied only for your own use."*

My evangelism within the Commodore User's Group of Springfield seems to be

More ►

responsible for quite a number of copies from the ten or twelve COMAL disks I placed in the club library. At least one member has subscribed to *COMAL Today*, and I imagine he and I will be ordering books for others.

The following program (written to introduce COMAL to the Commodore Users Group of Springfield) consists of two procedures which you can call from direct mode, just like adding commands to COMAL. It runs without change with COMAL 0.14 and 2.0 as well as with IBM PC COMAL. To use this program, type it in and RUN it. (Yes, nothing happened - yet!) Now, enter a direct command asking for a payment. For example, for principal of $1000, interest of 12.5%, and term of 24 months you type:

payment(1000, 12.5, 24)

The computer prints this:

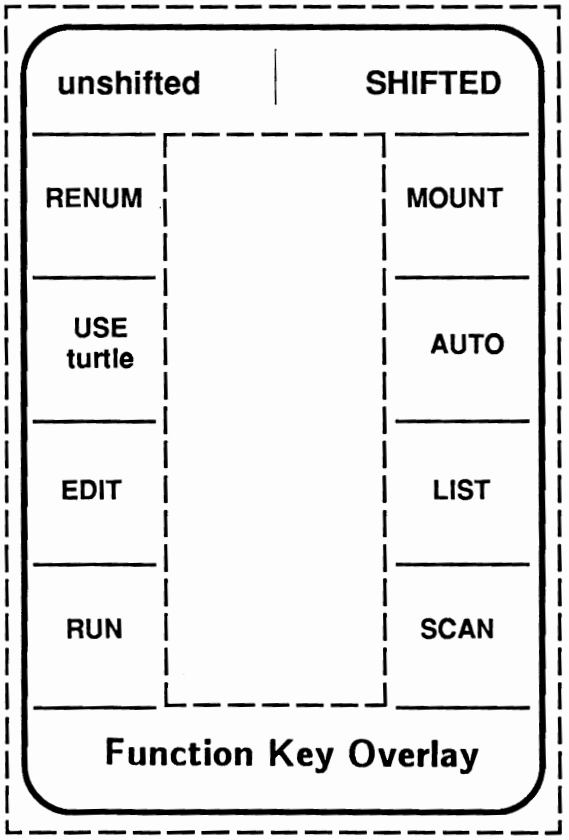
```
Monthly payment: $     47.31

Amount borrowed: $    1000.00
Total interest:  $     135.38
                      ----------
Total payback:   $    1135.38
```

[All Today Disks and User Group Disks plus the Auto Run Demo, Tutorial, Bricks Tutorial, Best of COMAL, Slide Show, Sampler and Paradise disks may be copied and placed in your local Users Group disk library. Don't pass around disks from our book/ disk sets or our specialty disks such as the Font or Games disk. Optional matching disks to books should be given only to others who also own that book. The four Cartridge Demo Disks may also be copied.] ■

Loan Payments

```
//Monthly payment computation, based on principal of loan,
//annual percentage rate of loan, and
//number of months over which loan will be paid back.
//  "parameters" are in parentheses in line below:
proc payment(principal,rate,months)
  if rate>1 then // Note multiple conditions in "if" test.
    rate:=rate/1200
  else  // If the "if" above is not true then...
    rate:=rate/12
  endif
  numerator:=principal*rate*(1+rate)^months
  denominator:=(1+rate)^months-1
  monthpay:=numerator/denominator
  print
  print
  print using "Monthly payment: $#####.##": monthpay
  summarize // Calling a procedure from within a procedure.
endproc payment
//
proc summarize
  payback:=months*monthpay
  interest:=payback-principal
  print
  print using "Amount borrowed: $#######.##": principal
  print using "Total interest:  $#######.##": interest
  print "                    ----------"
  print using "Total payback:   $#######.##": payback
endproc summarize
```

■



Okidata Package

by Terry Ricketts

Anyone who owns a printer other than the Commodore 1525 has learned, as I have, that CTRL-D (printscreen), is useless. Hence a number of packages for other printers have been written.

Since I own an Okidata 92 printer, I wrote one specific for it. But the code is general enough, with the printer specific code all placed in a table, that it can easily be adapted to other printers. I placed the package as high in memory as I could so as not to interfere with other packages. It has also been *rommed* so that it will stay in memory as new programs are loaded. *[Editors Note: The package has been put on Today Disk #12 in both rommed and non-rommed form. The names are pkg.oki92 and pkg.oki92-rommed.]*

When you give the command:

graphicscreen(0)

a '0' is stored in location $0c by COMAL to indicate that the resolution is Hi-Res. Similarly the command:

graphicscreen(1)

will put a '1' there. The package examines that location and switches to the appropriate screendump routine. A Hi-res screen will result in a screendump similar to many of the others that have been written for other printers. The picture is printed double size and sideways and takes up a full page of paper (is close to the size of the screen itself).

If a Multi-color screen is sensed, a routine is entered which converts the colors into a pattern of gray-scales, with each color having a unique pattern. This makes screendumps of color screens much more viewable. This code becomes more complex since each pair of bits represents one color. The pair must be decoded to one of 4 locations that contain the actual color information; the background color, the high and low nibbles of the character screen (which is located in the memory under the I/O field in $D800), and the character color located at $D800 in the I/O field. Each of these will provide a 4 bit number will decode to the 16 possible colors. This 4 bit number is used to index to a table of patterns for the colors.

Each multi-color dot will be printed as two rows of 4 bits each. This allows us to encode the color patterns as follows:

black	white	red	cyan
xxxx	oooo	xxxx	xoxo
xxxx	oooo	oooo	oxox

purple	green	blue	yellow
oxox	oxxo	xoxx	ooox
oxox	xoox	oxox	oxoo

orange	brown	lt red	grey1
oxox	oxxo	xoxo	xoxx
xooo	oxxx	oooo	oxxo

gray2	lt green	lt blue	gray3
oxxo	ooxo	xooo	oooo
xxxo	oxoo	ooox	xoox

Because of a design quirk in the Okidata 92 it can only print 7 dots per column when in graphics mode. This does not make printing bytes very easy. I decided that, even though it is slower, the code would be much easier if only 4 dots were printed in each pass. So a command has

More ►

to be sent to the printer to change the line feed to 8/144 inch per line which is equivalent to 4 dots. In addition the left margin was moved in so as to better center the picture on the page.

All of the printer specific commands are contained in a table starting at $BE66. There are 6 command strings stored there with each one alloted 8 bytes. Thus the commands could be changed for a different printer by re-assembling the source code or by merly poking the new values into memory. The commands are:

1) $BE66 Reset the printer
2) $BE6E Move the left margin
3) $BE76 Set the LF distance
4) $BE7E Turn graphics on
5) $BE86 Turn graphics off
6) $BE8E Send end of line return

In addition the string used to open the printer is located at $BE52. It is presently set to "U4:/S5". This will select device 4 and set most interfaces to transparent mode. If your interface is different you may need to patch that code.

The Hi-Resolution screen does allow you to put some color in, though not with as much selection. This package will not translate those colors. I considered doing that, but decided to put it off till later,(like most other things), since a number of complications arise when deciding how to handle the background colors.

I hope this package will prove as useful to others as it has to me. Feel free to use and modify as you wish.■

Roads To Rome

by Joe Visser

Working with interpretive languages usually leads to writing and conducting your algorithm behind the keyboard. If your algorithm doesn't work, you just fill in something else until it works. Because a procedure can be tested the moment it is fed into the computer, most people don't think about the best way a problem can be solved at the moment that a working procedure is devised to do it. Whether a solution is a good one or not is then virtually unimportant. The tedious business of constructing a program demands that a procedure is worked out on paper before translating it to a computer program.

Let's say we've got the following problem:

Write a program which translates a decimal number into an equivalent number in a numeric base between 1 and 37.

Writing the program to do this can be simplified to a procedure which does the conversion if we first write the body of the program:

```
PRINT "Conversion of a number."
DIM res$ OF 80 // This string will
//contain the conversion result
flag:=TRUE //incorrect input flag
REPEAT
  PRINT
  IF flag=FALSE THEN
    PRINT "Error! Again"
  ENDIF
  INPUT "Number  ?": number
  INPUT "In Base ?": base#
  flag:=FALSE
UNTIL base#>1 AND base#<37
```

More ►

```
IF number<0 THEN
  number:=ABS(number)
  res$:="-"
ENDIF
convert(number,base#,res$)
PRINT
PRINT res$
```

Due to the structured use of procedures our problem is now minimized to the conversion itself. The conversion procedure won't interfere with the main program as long as it is written neatly.

To convert a number to another base we first take a look at a number in the decimal base, e.g. "2314". If we number the digits starting at the right most digit with zero and then to the left with 1, 2, 3, etc. The number can be written:

2*10^3 + 3*10^2 + 1*10^1 + 4*10^0

or

2*1000 + 3*100 + 1*10 + 4*1 =2314

Other numeric bases work the same but use another number instead of 10.

To place this number in a string we first divide by 1000, giving 2. The result will become 2314-2*1000=314. Then divide that by 100, giving 3. We keep on dividing until we reach the power 0. The number will then be converted to a string.

So, to convert a number to a base n we must know the highest power present in that number. This can be calculated with the help of the LOG function. The integer part of the n-LOG(x) gives the highest power present in the number x in base n. The n-LOG(x) can be rewritten to LOG(x)/LOG(n) to accord with the LOG function built into your computer.

Our procedure looks like this:

```
PROC convert(number,ref base#,ref res$)
CLOSED // wrap line
  IF number<>0 THEN
    power:=INT(LOG(number)/LOG(base#))
    FOR count:=power TO 0 STEP -1 DO
      digit:=INT(number/(base#^count))
      number:=number-digit*base#^count
      IF digit<10 THEN
        res$:+CHR$(digit+ORD("0"))
      ELSE
        res$:+CHR$(digit-10+ORD("A"))
      ENDIF
    ENDFOR count
  ELSE
    res$:="0"
  ENDIF
ENDPROC convert
```

Our conversion procedure works! However, it isn't a procedure to be proud of. Further thinking leads us to the idea that instead of calculating the highest power present you can also start with the lowest power and then work your way up until you have nothing to convert any longer.

Our number 2314 would then be divided by 10 giving 231 as the integer part. The digit would then be 2314-231*10=4.

With 231 we do the same until nothing is left. The disadvantage of this method is that the digits are calculated in the wrong order. The most right digit is calculated first, the most left digit last. One way of getting around this is putting them in a string and after the conversion revert the complete string.

You can also place digits upon a stack

More ►

and later pop them off. Due to the LIFO structure of the stack, the first digit calculated is the last digit popped off. The second version of our procedure is:

```
PROC convert(number,ref base#,ref res$)
CLOSED // wrap line
  DIM stack(80)
  sp:=1 // This is the stack pointer
  REPEAT
    temp:=number MOD base#
    number:=number DIV base#
    stack(sp):=temp
    sp:+1
  UNTIL number=0
  FOR count:=sp-1 TO 1 STEP -1 DO
    digit:=stack(count)
    IF digit<10 THEN
      res$:+CHR$(digit+ORD("0"))
    ELSE
      res$:+CHR$(digit-10+ORD("A"))
    ENDIF
  ENDFOR count
ENDPROC convert
```

In this procedure the part that does the conversion is shorter than the part which places the digits in res$.

The method used in the second procedure gives us a hint about a recursive definition of the procedure. You must realize that the second procedure calculates a digit and then converts the remaining number. At this point, the recursion becomes obvious.

The third definition of the procedure:

```
PROC convert(number,ref base#,ref res$)
CLOSED // wrap line
  temp:=number MOD base#
  number:=number DIV base#
  IF number>0 THEN convert(number)
  IF temp<10 THEN
    res$:+CHR$(temp+ORD("0"))
  ELSE
    res$:+CHR$(temp-10+ORD("A"))
  ENDIF
ENDPROC convert
```

In this recursive definition of the conversion procedure, the computer controls the stack, while in the previous definition the user had to control the stack himself.

Now that we have conducted three methods of solving this problem, we can ask which of the three is best? I think there is a very simple answer to that question. The solution YOU understand is the best. Most people tend to forget that you must UNDERSTAND your own programs instead of borrowing very complicated procedures. ■

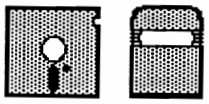
MORE CONVERSIONS

Someone else sent us the following conversion function that will return the decimal equivalent of any number represented in any base up to base 36. For example, to print the decimal number for hex $ff use:

```
PRINT DECIMAL("ff",16)

FUNC decimal(number$,base) CLOSED
  DIM n$ OF 36
  n$="0123456789abcdefghijklmnopqrstuvwxyz"
  l'n:=LEN(number$)
  IF l'n=1 THEN
    RETURN (number$ IN n$)-1
  ELSE
    rest:=decimal(number$(1:l'n-1),base)*base
    RETURN rest+decimal(number$(l'n),base)
  ENDIF
ENDFUNC decimal
```

■

Instructional Videos

by Garrett A. Hughes 

As a teacher, I have found that creating lessons and recording them on video tape is one of the most exciting ways to utilize C64 COMAL. The *"instructional videos"* that I produce are intended for viewing by my student audience.

To make a lesson, I use a standard video cassette recorder, a microphone, and a light pen in conjunction with C64 COMAL 2.0. While making a lesson, I employ the monitor screen as a *"blackboard"*. Instead of writing on the blackboard with chalk, I type to the screen from the keyboard, and draw on the screen with the lightpen. As I draw or type, I describe what I'm doing as if I were teaching a lesson to one of my classes.

Recording the lesson on video tape is easy. You need only connect the RF output on the back of the C-64 to the RF input on your video recorder. Connect your microphone to the audio input of the same VCR (use an audio amplifier if necessary). Start the recorder when you begin the lesson and presto, you have a permanent record of whatever you desire to teach.

The video tapes you make can be duplicated and kept in the school library. They can be checked out overnight like books on reserve or can be used in your school's audio-visual center. Students can use them to catch up on work they have missed, review material they didn't quite understand in class, or take a peek at more advanced material. Substitute teachers can use the tapes in your classes while you're away.

And that's just the beginning. Suppose a video lesson deals with the use of a piece of software that runs on the C64. First, load that software into the C64, then have the student play your video tape with the tape machine's outputs leading to the C64's monitor. Hitch the output of the C64 to the input of the VCR as described earlier. When the student wants to hear and see the lesson, he/she presses the PLAY button on the VCR. When the student wants to try what they are observing, they press the STOP button on the VCR. The same software that you are describing will now be lighting up the monitor screen. Yes, really! **Just pressing the PLAY or STOP button on the VCR shifts the student from a passive observer to an active participant or vice versa.**

When making a video, I use text and graphics screens that I have prepared ahead of time and stored on disk. The COMAL PACKAGES make it a cinch to prepare just the screen I want, and to retrieve it at the appropriate time.

With COMAL's lightpen and sprite packages, I can create images that move to any location on the screen to which I point. The effect is striking and a minimum of programming effort is involved. For example, suppose you have a graphics picture you want to talk about. You can create a sprite *"pointer"* and move the pointer about the screen under lightpen control. You can put the pointer wherever you want on the graphic image. The video viewer is directed by the pointer to the exact location on the screen that you are talking about.

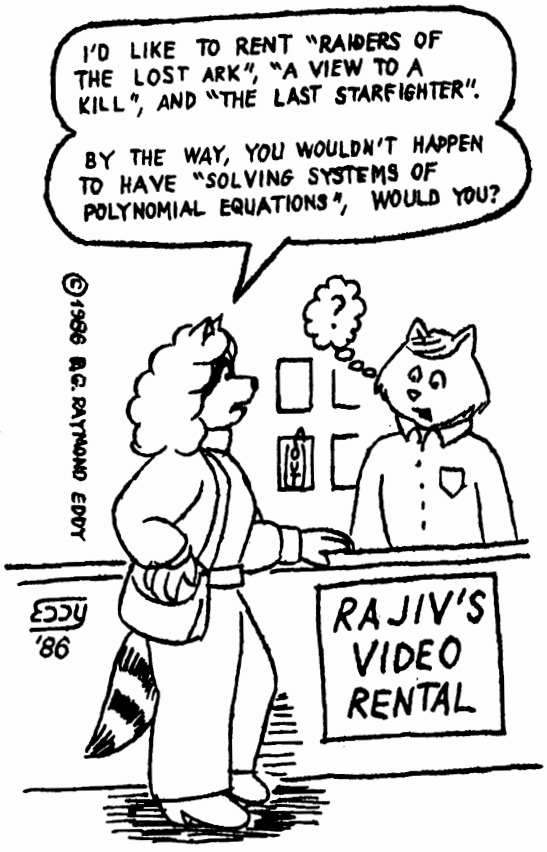
The best feature of instructional videos, which you prepare yourself, is that they allow you to take advantage of your own years of teaching experience.

More ►

That's something that a lesson under program control, or a commercially prepared videodisk can never duplicate. Another advantage of this system is that the price of all the components is affordable, and many of your students will already have access to a VCR.

Give it a try. I can almost guarantee that the response to your efforts by your students and school officials will be overwhelmingly positive.

[Ed. Note: This is a fabulous method for preparing lessons! The same techniques could be used for COMAL 0.14 programs, or any other language. Garrett teaches at Burlington High School in Burlington, Vermont 05401]. ■



VAL and STR$ 0.14

by Dick Klingens

Many different procedures and functions have been written to emulate the VAL and STR$ commands left out of COMAL 0.14. Each of these have either been simple and short (not doing fractional numbers or handling negative numbers) or complex and long (taking up more memory).

The following two procedures work using any Commodore compatible disk drive (1541, MSD, etc.), provide full conversion of numbers and strings, and take up little space.

Each routine uses a *"buffer"* in the disk drive's memory. First the buffer is opened and the number or string is printed to it. Then the buffer pointer is reset to the beginning and the number or string is read back in (in the desired format). The final step is to close the file and return the result. There doesn't even have to be a disk in the drive (although the drive does have to be turned on).

```
func val(x$) closed
  open file 100,"#",unit 8,2,read
  print file 100: x$ // print to file
  pass "b-p:2,1" // reset to beginning
  input file 100: y // bring back in
  close file 100 // new form and close
  return y
endfunc val
//
proc str(x,ref y$) closed
  open file 100,"#",unit 8,2,read
  print file 100: x // print to file
  pass "b-p:2,1" // reset to beginning
  input file 100: y$ // bring back in
  close file 100 // new form and close
endproc str
```
■

Fractals

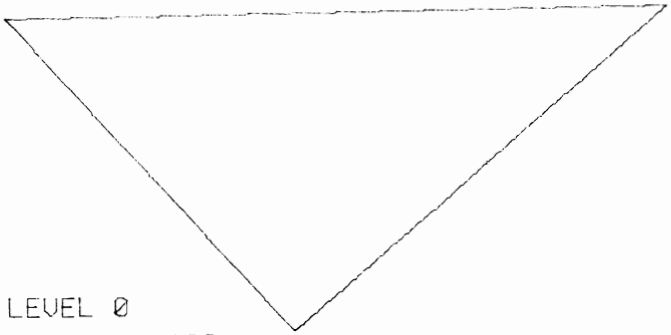
by Kevin Quiggle 

This program uses the 1520 plotter or the graphics screen to display fractal images. The plotter display is much more impressive.

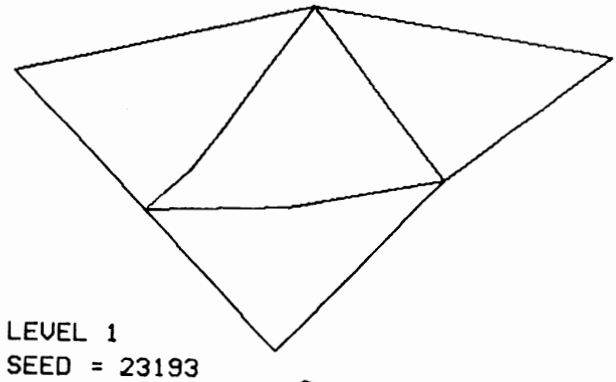
Three dimensional fractal diagrams are drawn at up to seven levels of complexity. You get to choose the complexity level and the seed (for the random number generator). If you use the same seed twice, you should get the same results each time. A different seed should give a different fractal diagram. If you don't enter a seed, the program will select one for you.

When the display if finished, the plotting time is also displayed. It takes nearly 4 1/2 hours to plot a level 7 fractal on the 1520 plotter.

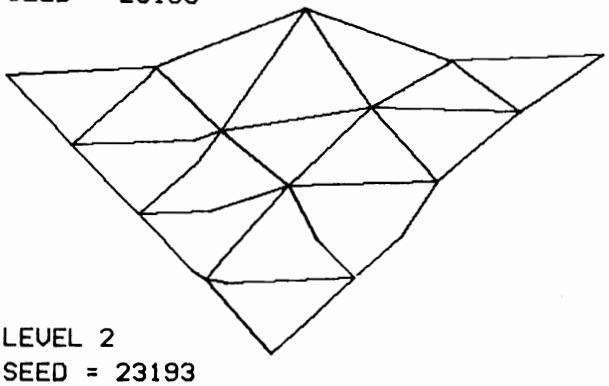
Fractals are a very interesting subject. They are a sophisticated way to generate 3D graphics. It has been reported that Lucasfilm has used fractals in every computer game they've produced (the Epyx game *Koronis Rift* for example). Fractals were also used in the second *Star Trek* movie. Ask at your local library for some reference articles about fractals. Scientific American had an in depth article a few years ago.



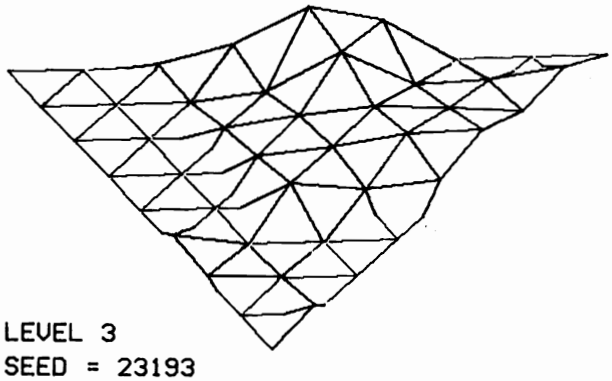
LEVEL 0
SEED = 23193



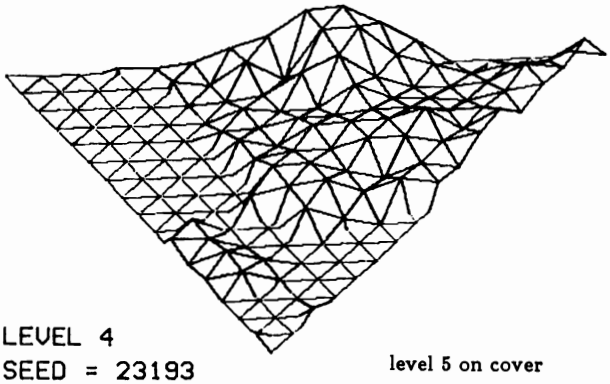
LEVEL 1
SEED = 23193



LEVEL 2
SEED = 23193



LEVEL 3
SEED = 23193



LEVEL 4
SEED = 23193

level 5 on cover

More ►

```
// delete "3d'fractals"
// save   "3d'fractals"
// by kevin quiggle
//
main
//
PROC main
 PAGE
 ask_level
 init
 ask_seed
 select_output //screen or plotter
 FOR n#:=1 TO level# DO
  l#:=10000/1.8^n#
  PRINT "Working on level ",n#
  ib#:=mx#/2^n#; sk:=ib#*2
  assign_x_hts
  assign_y_hts
  assign_diag_hts
 ENDFOR n#
 set_scale
 init_graphics
 show_level
 settime("0")
 plot_result
 show_plot_time
ENDPROC main
//
PROC select_output
 PRINT
 PRINT "Select screen or plotter (s/p): ",
 REPEAT a$:=KEY$ UNTIL a$ IN "sSpP"
 PRINT a$
 IF a$ IN "pP" THEN device#:=plotter#
 PRINT
ENDPROC select_output
//
PROC ask_level
 PRINT "Number of levels (0-7): ",
 REPEAT a$:=KEY$ UNTIL a$ IN "01234567"
 PRINT a$
 level#:=VAL(a$)
ENDPROC ask_level
//
PROC init
 USE system
 DIM dat#(0:128,0:64)
 ds#:=2
 FOR n#:=1 TO level# DO ds#:+2^(n#-1)
 mx#:=ds#-1; my#:=mx#/2; rh:=PI*30/180
 vt:=rh*1.2; screen#:=2 plotter#:=4
 device#:=2 //default is screen
 sea:=6 //default color for sea
 land:=0 //default color for land
 z2:=0; x2:=0; y2:=0
ENDPROC init
//
PROC ask_seed
 REPEAT
  INPUT AT 3,0: "Enter a seed value (1-32767): ": a$
  IF a$<>"" THEN
   TRAP
    seed#:=VAL(a$)
    IF seed#>0 THEN a$:="ok"
   HANDLER
    PRINT AT 5,0: "Please enter a number from 1 to 32767"
   ENDTRAP
  ELSE
   seed#:=RND(1,32768)
  ENDIF
 UNTIL a$="ok" OR a$=""
 RANDOMIZE seed#
ENDPROC ask_seed
//
PROC assign_x_hts
 FOR ye:=0 TO mx#-1 STEP sk DO
  FOR xe:=ib#+ye TO mx# STEP sk DO
   ax#:=xe-ib#; ay#:=ye
   get_data
   d1#:=d#; ax#:=xe+ib#
   get_data
   d2#:=d#
   d#:=(d1#+d2#)/2+RND(1,l#/2)-l#/4
   ax#:=xe; ay#:=ye
   put_data
  ENDFOR xe
 ENDFOR ye
ENDPROC assign_x_hts
//
PROC assign_y_hts
 FOR xe:=mx# TO 1 STEP -sk DO
  FOR ye:=ib# TO xe STEP sk DO
   ax#:=xe; ay#:=ye+ib#
   get_data
   d1#:=d#; ay#:=ye-ib#
   get_data
   d2#:=d#
   d#:=(d#+d2#)/2+RND(1,l#/2)-l#/4
   ax#:=xe; ay#:=ye
   put_data
  ENDFOR ye
 ENDFOR xe
ENDPROC assign_y_hts
//
PROC assign_diag_hts
 FOR xe:=0 TO mx#-1 STEP sk DO
  FOR ye:=ib# TO mx#-xe STEP sk DO
   ax#:=xe+ye-ib#; ay#:=ye-ib#
   get_data
   d1#:=d#; ax#:=xe+ye+ib#
   ay#:=ye+ib#
   get_data
   d2#:=d#; ax#:=xe+ye
   ay#:=ye
   d#:=(d1#+d2#)/2+RND(1,l#/2)-l#/4
   put_data
  ENDFOR ye
 ENDFOR xe
ENDPROC assign_diag_hts
//
PROC get_data
 IF ay#>my# THEN
  by#:=mx#+1-ay#
  bx#:=mx#-ax#
 ELSE
  by#:=ay#
```

More ►

```
    bx#:=ax#
   ENDIF
   d#:=dat#(bx#,by#)
 ENDPROC get_data
 //
 PROC put_data
  IF ay#>my# THEN
    by#:=mx#+1-ay#
    bx#:=mx#-ax#
  ELSE
    by#:=ay#; bx#:=ax#
  ENDIF
  dat#(bx#,by#):=d#
 ENDPROC put_data
 //
 PROC set_scale
  xs:=.04; ys:=.04; zs:=.04
 ENDPROC set_scale
 //
 PROC show_level
  CASE device# OF
  WHEN screen#
   plottext(0,-185,"Level "+STR$(level#))
   plottext(0,-200,"seed = "+STR$(seed#))
  WHEN plotter#
   p_moveto(0,-300)
   p_char("LEVEL "+STR$(level#))
   p_moveto(0,-325)
   p_char("SEED = "+STR$(seed#))
  ENDCASE
 ENDPROC show_level
 //
 PROC show_plot_time
  CASE device# OF
  WHEN screen#
   plottext(0,-225,"Plot Time "+gettime$)
   plottext(0,-250,"Hit STOP to quit.")
   TRAP ESC-
   REPEAT UNTIL ESC
   TRAP ESC+
   textscreen
  WHEN plotter#
   p_moveto(0,-350)
   p_char("Plot Time = "+gettime$)
   p_moveto(0,-400)
   p_reset
  ENDCASE
 ENDPROC show_plot_time
 //
 PROC plot_result
  FOR ax#:=0 TO mx# DO
   xo:=-999
   FOR ay#:=0 TO ax# DO
    get_data
    zz:=d#
    yy:=ay#/mx#*10000
    xx:=ax#/mx#*10000-yy/2
    check_sea_level
    move_or_plot
   ENDFOR ay#
  ENDFOR ax#
  FOR ay#:=0 TO mx# DO
   xo:=-999
   FOR ax#:=ay# TO mx# DO
    get_data
    zz:=d#
    yy:=ay#/mx#*10000
    xx:=ax#/mx#*10000-yy/2
    check_sea_level
    move_or_plot
   ENDFOR ax#
  ENDFOR ay#
  FOR ex#:=0 TO mx# DO
   xo:=-999
   FOR ey#:=0 TO mx#-ex# DO
    ax#:=ex#+ey#; ay#:=ey#
    get_data
    zz:=d#
    yy:=ay#/mx#*10000
    xx:=ax#/mx#*10000-yy/2
    check_sea_level
    move_or_plot
   ENDFOR ey#
  ENDFOR ex#
 ENDPROC plot_result
 //
 PROC check_sea_level
  IF level#<>0 THEN
   IF xo=-999 THEN
    IF zz<0 THEN
     set_color(sea)
     z2:=zz; zz:=0
    ELSE
     set_color(land)
     z2:=zz
    ENDIF
   ELSE
    IF z2>0 AND zz>0 THEN
     z2:=zz
    ELSE
     IF z2<0 AND zz<0 THEN
      z2:=zz; zz:=0
     ELSE
      w3:=zz/(zz-z2); x3:=(x2-xx)*w3+xx
      y3:=(y2-yy)*w3+yy
      z3:=0; zt:=zz; yt:=yy; xt:=xx
      IF zz<=0 THEN //water level
       zz:=z3; yy:=y3; xx:=x3
       move_or_plot
       set_color(sea)
       zz:=0; yy:=yt
       xx:=xt; z2:=zt
      ELSE
       zz:=z3; yy:=y3; xx:=x3
       move_or_plot
       set_color(land)
       zz:=zt; yy:=yt; xx:=xt
       z2:=zz
      ENDIF
     ENDIF
    ENDIF
   ENDIF
   x2:=xx; y2:=yy
  ENDIF
 ENDPROC check_sea_level
 //
 PROC move_or_plot
  xx:=xx*xs; yy:=yy*ys
  zz:=zz*zs
  rotate
  tilt_up
  IF xo=-999 THEN
   pr$:="m"
  ELSE
   pr$:="d"
  ENDIF
  xp:=INT(yy) //+cx#
  yp:=INT(zz)
  IF xo=-999 THEN
   move_to
  ELSE
   draw_to
  ENDIF
  xo:=xp
 ENDPROC move_or_plot
 //
 PROC rotate
  IF xx<>0 THEN
   ra:=ATN(yy/xx)
   IF xx<0 THEN ra:+PI
  ELSE
   IF yy<0 THEN
    ra:=-PI/2
   ELSE
    ra:=PI/2
   ENDIF
  ENDIF
  r1:=ra+rh
  rd:=SQR(xx*xx+yy*yy)
  xx:=rd*COS(r1)
  yy:=rd*SIN(r1)
 ENDPROC rotate
 //
 PROC tilt_up
  rd:=SQR(zz*zz+xx*xx)
  IF xx=0 THEN
   ra:=PI/2
  ELSE
   ra:=ATN(zz/xx)
   IF xx<0 THEN ra:+PI
  ENDIF
  r1:=ra-vt
  xx:=rd*COS(r1)+xx
  zz:=rd*SIN(r1)
 ENDPROC tilt_up
 //
 PROC move_to
  CASE device# OF
  WHEN screen#
   moveto(xp,yp)
  WHEN plotter#
   p_moveto(xp,yp)
  ENDCASE
 ENDPROC move_to
 //
 PROC draw_to
  CASE device# OF
  WHEN screen#
   drawto(xp,yp)
```

More ►

```
  WHEN plotter#
    p_drawto(xp,yp)
  ENDCASE
ENDPROC draw_to
//
PROC set_color(color)
  CASE device# OF
  WHEN screen#
    pencolor(color)
  WHEN plotter#
    p_color(color)
  ENDCASE
ENDPROC set_color
//
PROC init_graphics
  CASE device# OF
  WHEN screen#
    USE graphics
    graphicscreen(0)
    window(-10,450,-250,100)
    fullscreen
  WHEN plotter#
    TRAP
      p_moveto(0,-100)
    HANDLER
      SELECT OUTPUT "ds:"
      STOP "Turn on Plotter!"
    ENDTRAP
    p_init
    land:=0 //black land color
    sea:=1 //blue sea color
  ENDCASE
ENDPROC init_graphics
//
//save "1520driver
// by kevin quiggle
//
PROC p_open(sa) CLOSED
  SELECT "u6:/s"+STR$(sa)
ENDPROC p_open
//
PROC p_close CLOSED
  SELECT OUTPUT "ds:"
ENDPROC p_close
//
PROC p_char(c$)
  p_open(6)
  PRINT 1
  p_open(0)
  IF c$<>"" THEN
    PRINT c$,
  ELSE
    PRINT c$
  ENDIF
  p_close
ENDPROC p_char
//
PROC p_home
  p_open(1)
  PRINT "h"
  p_close
ENDPROC p_home
//
PROC p_init
  p_open(1)
  PRINT "i"
  p_close
ENDPROC p_init
//
PROC p_moveto(x,y)
  p_open(1)
  PRINT "m";x;""29"";
  PRINT y;""29""
  p_close
ENDPROC p_moveto
//
PROC p_drawto(x,y)
  p_open(1)
  PRINT "d";x;""29"";
  PRINT y;""29""
  p_close
ENDPROC p_drawto
//
PROC p_move(x,y)
  p_open(1)
  PRINT "r";x;""29"";
  PRINT y;""29""
  p_close
ENDPROC p_move
//
PROC p_draw(x,y)
  p_open(1)
  PRINT "j";x;""29"";
  PRINT y;""29""
  p_close
ENDPROC p_draw
//
PROC p_reset
  p_open(7)
  PRINT
  p_close
ENDPROC p_reset
//
PROC p_color(color)
  p_open(2)
  PRINT color
  p_close
ENDPROC p_color
//
PROC p_charsize(size)
  p_open(3)
  PRINT size
  p_close
ENDPROC p_charsize
//
PROC p_rotchar(rot)
  p_open(4)
  PRINT rot
  p_close
ENDPROC p_rotchar
//
PROC p_scribe(brk)
  p_open(5)
  PRINT brk
  p_close
ENDPROC p_scribe
```

QLink Simulator

by Captain COMAL

Over a year ago, we began providing COMAL support via the online network PlayNet. Recently we have added Quantum Link as one of our official support networks (see their ad on page 9). So you can *"see"* what it's like, we wrote a QLink Simulator (on *Today Disk #12*). If you decide to try QLink, we have a national COMAL meeting every second Thursday of each month at 10pm Eastern time. We also maintain an active Question & Answer board. It is inside the Commodore Information Network under Meet The Press. If you have a question about COMAL, just post it on our Q&A board, and usually within a day or two you will have an answer. In addition, we are uploading important COMAL programs into our section inside Meet The Press.

The QLink simulator is nice because you can get the *"feel"* of it without using a modem - any time of the day. You can easily customize the data statements near the end of the program. There is about 2K free memory space to hold extra "messages". Have you and your friends be the ones *"talking"*.

The QLink simulator makes use of a "bug" of COMAL 0.14. The **background** command does not change the entire graphics screen background color (as in COMAL 2.0), only the drawing background color from that point on. This allows text to be put on the graphic screen with separate background colors. QLink does this with complex interrupt timing controls. Our simulator does it with one statement!

If you use QLink, you can contact us by EMAIL. David Stidolph is COMALite D and Len Lindsay is Captain C. ■

Ralfs COMAL Corner

by Ralph McMullen 

This article is about using files with COMAL 0.14. Except as noted, everything I say about this version of COMAL works with the cartridge version 2.0 as far as the file handling I'll be talking about. The cartridge version has some more bells and whistles yet, but this column is currently a get-started affair for those who haven't plunged for the cartridge. Incidentally, the cartridge is currently a bargain at under $100, with documintation!

Sequential Files

Sequential files are slightly less clumsy to use than in BASIC. For instance, to open a sequential file to write, use:

open file 6,"testfile",write

The 6 can be any number, theoretically 1-255. The system uses some of these numbers. You can stay out of trouble if you don't use numbers less than 2 nor greater than 253.

Similiarly, "testfile" can be any name you like, the quotes are required. Instead of **write**, you can say **read** or **append**. They work just like you'd think. Remember: **read** reads from; **write** writes over; **append** adds to.

Some examples of usage after the open statement are:

read file 5: x,y,z
write file 9: text$

The next examples are for COMAL 2.0 only, and read/ write a whole array at a time. In the last case, an entire array of string variables, each 20 characters long, is written to the disk file by one statement.

```
dim array (2,5)
open file 5,"array.data",read
read file 5: array
close file 5

dim name$(8) of 20
open file 6,"names.data",write
write file 6: name$
close file 6
```

To append a string variable, two real variables, and an integer variable to the end of the file defined as 6 in the open statement:

```
open file 6,"example",append
write file 6: word$, xray, yoyo, int%
```

When done using a file:

close file 6

Random/Direct Access Files

It is in the handling of random or direct access files that COMAL really excells. To open the direct access file, just type:

open file 9,"testfile",random 132

This opens file 9 with a name of "testfile" and a record length of 132 bytes. If you didn't want to ask, a byte is eight bits, and is the amount of memory needed to store one ASCII character.

What's good about a random file?

You don't have to go serially through all the records to get to the one you

More ►

want. If you know which record you want, you can go directly to it. Hence "direct access".

There are two things you need to know. First, when you first open or create the file, you need to know what record length to use. Second, when you go to access a record, you need to know the record number of the record you want.

The first is just a matter of counting and adding, we can have the computer tell us the seconds.

How long is a record?

The limit is 254 bytes. You will want to make the record as short as you can, and still hold all you data. The reason is that the full length of the record is stored on disk, even if you don't fill it up.

It all adds up!

The way it does this is:

For each number, whether integer or decimal, allow 5 bytes. (In COMAL 2.0 its 5 bytes for decimal numbers and 2 bytes for integers). For each string count the number of characters in the string, and add two.

For Example:

Suppose you wanted to keep a simple checkbook register. You want to keep track of the check number, who it's to, what it's for, and the amount. In real life, you'd probably want to keep track of deposits also. The simple way is to just add another number field, so you can have an amount paid, and an amount deposited. The who to and what for fields can each be about 20 characters, so the record length adds up as follows:

Bytes	Items
5	Check no#
22	Who to
22	What for
5	Amount paid
5	Amount deposited
===	
59	Total

So the record length for a check register file would be 59 bytes.

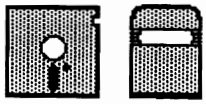
You can get started filling in your check register, using something like this:

```
dim who$ of 20
dim what$ of 20
open file 9,"register",random 59
index:=1
repeat
 input "check no: (0 to quit)":check #
 if check#<>0 then
  input "who to:" who$
  input "what for:": what$
  input "amt. paid:": paid
  input "amt. depos.:": deposit
  write file 9,index: check#,who$,what$,
  paid,deposit // wrap line
  index:+1
 endif
until check# = 0
close file 9
```

Note that check# is an integer variable. This saves space in memory (also on disk in COMAL 2.0). If you are using COMAL 2.0, remember to adjust your record length accordingly. COMAL 0.14 writes integers as two bytes followed by 3 zero bytes, which makes them take up 5 bytes.

[reprinted from the TCCC Newsletter] ■

Pic Finder

by Colin Thompson 

One of the most frustrating things about working with "Picture Files" is finding pictures to work with. If you use any of the commercially available "paint" programs like Doodle or Koala, you should have no shortage of picture files. If you don't have one of these programs, you may have to put on your detective's hat and search them out.

To assist you with your search, I've written *Pic Finder*. The object of *Pic Finder* is to examine disks, looking for picture files made by a variety of Paint programs. *Pic Finder* can identify picture files made made by the following programs and methods:

Doodle	Koala
Animation Station	Blazing Paddles
Super Sketch	Scribbler
Print Shop (3 blk)	Flexidraw 5.0
Paint Magic	Video Basic
BASIC Bitmaps	COMAL Bitmaps
Compact Bitmaps	Color Compact
COMAL hi-res color	COMAL multicolor

In addition, *Pic Finder* can even locate some files that have been renamed. Files made with Doodle, Koala, Blazing Paddles, and Animation Station will be found, even if they have been renamed (no longer follow the naming convention employed by the creating program). This is a pretty handy feature because these files are frequently renamed when used in commercial game programs. Once renamed, they are difficult to identify without *Pic Finder*. All of the picture files listed above may be converted for use in the COMAL environment.

Using *Pic Finder* is fun and revealing. LOAD the program, then drag out every disk you own. Start with your game disks. They sometimes use Doodle or Koala pictures for game screens. RUN the *Pic Finder* program and follow the simple instructions on the screen. Simply insert the disk to be examined and press the appropriate key. When *Pic Finder* finds a picture file, all relevant information about that file will be printed on the screen, including the file's size in blocks, filename, filetype, and the kind of file it is. An **R*** shows that the file had been renamed.

Another source of picture files are the COMAL Today and User Group disks. Like most hackers, I have a never-ending supply of old BASIC program disks. Most of these come from Commodore user group libraries. These disks usually contain BASIC public domain programs that no self respecting COMALite would even look at. Surprisingly, they are a good source of picture files - especially BASIC bitmaps that may be converted easily for use in COMAL.

Once you've found some picture files, what do you do with them? Today Disks #9, #10, #11, and #12, and User Group #5, #10, and #11 have COMAL programs that manipulate picture files. Some of these programs convert files from one format to COMAL format, while others let you see the pictures on the screen or print them on your dot matrix printer. Some people just like to collect them for their artistic value. Whatever you decide to do with the pictures, *Pic Finder* will uncover and identify the files for you.

TECHNICAL DETAILS

Every source file has a unique combination of file size, file type,

More ►

load address, or filename convention. *Pic Finder* does it's detective work by examining the disk's directory. The routine used is a variation of George Jones' read'directory proc first published in *COMAL Today #7*. David Stidolph converted the proc to be used in COMAL 0.14.

The routine used in *Pic Finder*, examines each directory entry, looking first for a SEQ or PRG filetype. Next the filename is matched up in a CASE statement that knows all of the filename conventions. For example, Doodle files have "dd" as the first two characters in the filename. They are 37 block PRG files. If all three of these pieces of information match, the filename and other information is printed on the screen, and the next filename is examined. This is a fairly simple "pattern match" search. As long as the source filename has not been altered, the file can be accurately identified.

Some source files cannot be identified by filename convention. Examples include Paint Magic, Video Basic, BASIC Bitmaps, and Print Shop files. When *Pic Finder* encounters a filename that doesn't match any known convention, it looks at the file type and file size. If these match a partial description of a known source file, the filename is stored in an array for later examination.

After the first pass through the disk's directory, the list of "possible" names is passed to a different section of the program. This section finds the file's load address, the first two bytes in a PRG file. The recovered load address is passed through a case statement that can identify source files by matching load addresses, file sizes, and file types.

If a match is found, the name is printed on the screen.

This is also how renamed files can be identified. The technique may be used to identify almost any kind of disk file, not just picture files. The routine could be modified to form the basis of a fast directory printer in COMAL 0.14.

ADDITIONAL NOTES

Animation Station and Blazing Paddles share identical filename conventions and file structures. They have a "PI." prefix. Flexidraw 3.0 and 4.0 files will be identified as a "BASIC bitmap", even though the color file may be recorded separately on the disk.

Further Reference:

Graphics Editor System, COMAL Today #11, page 57
Bitmap - A New Package, COMAL Today #10, page 56
Load / Save Compact Bitmaps, COMAL Today #10, page 61
Bitmap Compression in COMAL, COMAL Today #10, page 68
Bitmaps in COMAL, COMAL Today #9, page 38
Color Pictures For COMAL 0.14, COMAL Today #6, page 66
Graphic Screens In COMAL 0.14, COMAL Today #6, page 75
Chinese Screens, COMAL Today #5, page 18
Savescreen Correction, COMAL Today #5, page 37
How To Dump a Graphic Screen To Commodore Printer, COMAL Today #4, page 30
Screen Dumps, COMAL Today #4, page 47
Save Graphics Screen, COMAL Today #3, page 38
Load Graphics Screen, COMAL Today #3, page 39

More ►

```
// delete "0:picfinder.14"
// by Colin Thompson
// save  "0:picfinder.14"
// update by Captain COMAL for
// scratch protected file types
setup
title
dir
if map then find'maps
pencolor (13)
print "+--------------------------------------+"
while key$=chr$(0) do null
//
proc title
 print chr$(14)
 pencolor (1)
 print chr$(18)+"          PICTURE FINDER",tab(40)
 pencolor (7)
 print chr$(18)+"       by Colin F. Thompson",tab(40)
 print
 print
 pencolor (1)
 print "    Assuming you are not familiar with "
 print "all the various kinds of picture files, "
 print "this program will examine your disk and "
 print "report to you all the ""pictures"" it can "
 print "identify.  It can find,  with certainty"
 print "most of the popular file formats.      "
 pencolor (13)
 print "    Any file identified with an"
 pencolor (10)
 print tab(6),chr$(18),"R*",chr$(18+128);
 pencolor (13)
 print "prefix has been renamed."
 print
 print
 pencolor (1)
 print chr$(18)+" Insert the source disk in the drive",tab(40)
 pencolor (7)
 print chr$(18)+"    Press the SPACE BAR to begin",tab(40)
 while key$<>chr$(32) do null
 pencolor (13)
 print chr$(147),
 print "+--------------------------------------+"
endproc title
//
proc dim'stmnts
 dim entry$ of 32
 dim name$ of 16
 dim type$ of 1
 dim syze$ of 1
 dim flag$ of 15
endproc dim'stmnts
//
proc check
 t:=ord(type$)
 s:=ord(syze$)
 flag$:=""
 if (t>128 and t<131) or (t>192 and t<195) then
  case t of
  when 129,193 // SEQ
   type$:="S"
   if entry$(4:7)="hrg." and s=40 then flag$:="COMAL multicol"
   if entry$(4:7)="hrg." and s=36 then flag$:="COMAL hi-res"
   if entry$(4:7)="scr." and s=33 then flag$:="Scribbler"
   if ".crg" in entry$ then flag$:="Compact bitmap"
   if "crg." in entry$ then flag$:="Compact color"
  when 130,194 // PRG
   type$:="P"
   if entry$(4:5)="dd" and s=37 then flag$:="Doodle"
   if entry$(4:5)="fd" and s=37 then flag$:="Flexidraw 5.0"
   if entry$(4:6)="pi." and s=41 then flag$:="AS/Blazing pads"
   if entry$(4)=chr$(129) and s=40 then flag$:="Super sketch"
   if entry$(4:8)=chr$(129)+"pic " and s=40 then flag$:="Koala
    pad" // wrap line
   if ".hrg" in entry$ and s=32 then flag$:="COMAL bitmap"
   if flag$="" then
    case s of
    when 3,32,33,37,40,41
     j:+1
     bit$(j):=entry$(4:19) //bitmap??
     siiz(j):=s
     map:=true
    otherwise
    endcase
   endif
  otherwise
  endcase
 endif
 if flag$>chr$(0) then update'screen
endproc check
//
proc update'screen
 pencolor (13)
 print "|",
 pencolor (9)
 if s>9 then
  print s,tab(5),
  pencolor (14)
 else
  print " ",s,tab(5),
  pencolor (14)
 endif
 if entry$(4)=chr$(129) then
  print chr$(18)+"A",chr$(146),
  entry$:=entry$(5:19)
 else
  case flag$ of
  when "BASIC bitmap","Paint magic","R*Doodle","R*Blazing P
  add","R*Koala" // wrap line
   entry$:=entry$
  when "Video BASIC","Printshop image"
   entry$:=entry$
  otherwise
   entry$:=entry$(4:19)
  endcase
 endif
 print entry$,tab(22),
 pencolor (13)
 print type$;
 pencolor (7)
 print flag$,tab(39),
 pencolor (13)
 print "|"
endproc update'screen
//
```

```
41 pi.colorwatch     P BLAZING PADDLES
33 scr:scribbler     P SCRIBBLER
40 hrg:red dragon    P COMAL MULTICOL
40 Whatgirl          P SUPER SKETCH
40 pic a bonanza     P KOALA PAD
37 ddbaseball cover  P DOODLE
37 fdpirate          P FLEXIDRAW 5.0
36 hrg.calvin        P COMAL HIRES
4  lightbulb.crs     P COMPACT BITMAP
10 crg.fake!!!       P COMPACT COLOR

32 a basic bitmap    P BASIC BITMAP
33 eclipse.graph     P VIDEO BASIC
37 two girls         P PAINT MAGIC
40 light house       P RMKoala
37 middle earth      P RMDoodle
41 worldmap          P RMBlazing Pads
3  cookie monster    P PRINTSHOP IMAGE
```

More ►

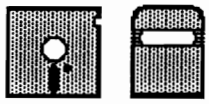
Pic Finder - continued

```
proc wait
 while key$<>chr$(0) do null
 while key$=chr$(0) do null
endproc wait
//
proc setup
 print chr$(147),
 background 0
 border 0
 dim yn$ of 1
 printit:=false
 c:=0 // load address
 k:=1 // color pic counter
 j:=0 // bitmap counter
 l:=0 // paint magic counter
 map:=false
 pm:=false
 dim bit$(20) of 60
 dim siiz(20)
endproc setup
//
proc find'maps
 pencolor (13)
 print "+-------------------------------------+"
 for i:=1 to j do
  c:=0
  comma:="," in bit$(i)
  if comma then bit$(i)(comma):="*"
  open file 10,"0:"+bit$(i)+",p",read
  a:=disk'get(10,file'end)
  b:=disk'get(10,file'end)
  close file 10
  c:=a+(b*256)
  case c of
  when 8192,16384,24576,32768,57344
   if siiz(i)=32 or siiz(i)=33 then
    entry$:=bit$(i)
    flag$:="BASIC bitmap"
    type$:="P"
    s:=siiz(i)
    update'screen
   else
    if c=24576 and siiz(i)=40 then
     entry$:=bit$(i)
     flag$:="R*Koala"
     type$:="P"
     s:=40
     update'screen
    endif
   endif
  when 22528
   if siiz(i)=3 then
    entry$:=bit$(i)
    flag$:="Printshop image"
    type$:="P"
    s:=3
    update'screen
   endif
  when 16270
   if siiz(i)=37 then
    entry$:=bit$(i)
    flag$:="Paint magic"
    type$:="P"
    s:=37
    update'screen
   endif
  when 40960
   if siiz(i)=41 then
    entry$:=bit$(i)
    flag$:="R*Blazing Padd"
    type$:="P"
    s:=41
    update'screen
   endif
   if siiz(i)=37 then
    entry$:=bit$(i)
    flag$:="Video BASIC"
    type$:="P"
    s:=37
    update'screen
   endif
  when 23552
   if siiz(i)=37 then
    entry$:=bit$(i)
    flag$:="R*Doodle"
    type$:="P"
    s:=37
    update'screen
   endif
  otherwise
  endcase
 endfor i
endproc find'maps
//
proc dir
 disk'get'init(2)
 disk'init
 dim block$ of 258, ds$ of 2
 sector:=1
 dim'stmnts
 pass "i0"
 open file 2,"#2",unit 8,2,read
 ds$:=status$
 if ds$="00" then
  repeat
   setblock(sector)
   get'entries
  until sector>18
 else
  close
  print "OOOPS! Something's wrong here."
  stop
 endif
 close file 2
endproc dir
//
proc get'entries
 track:=ord(block$(1))
 sector:=ord(block$(2))
 for pos:=3 to 227 step 32 do
  if block$(pos)>chr$(127) then
   type$:=block$(pos)
   syze$:=block$(pos+28)
   entry$:=block$(pos:pos+29)
   check
  endif
```

More ►

```
 endfor pos
endproc get'entries
//
proc setblock(sec)
 if sec>9 then
  pass "u1: 2 0 18 1"+chr$((sec mod 10)+48)
 else
  pass "u1: 2 0 18 "+chr$(sec+48)
 endif
 block$(1:256):=""
 if block$="" then null
 start:=peek(51)+peek(52)*256+4
 poke 51,start mod 256
 poke 52,start div 256
 sys 1000
endproc setblock
//
proc disk'get'init(f'num) closed
 a:=1000
 for i:=1 to 19 do
  read byte
  poke a,byte
  a:+1
 endfor i
 //
 data 162,2,32,198,255,160,0
 data 32,207,255,145,51,200,208
 data 248,32,204,255,96
endproc disk'get'init
//
func disk'get(file'num,ref file'end) closed
 poke 2026,file'num
 sys 2025
 file'end:=peek(144)
 return peek(2024)
endfunc disk'get
//
proc disk'init closed
 for loc#:=2024 to 2039 do
  read v
  poke loc#,v
 endfor loc#
 data 0,162,0,32,198,255,32,207
 data 255,141,232,7,32,204,255,96
endproc disk'init
```

Benchmark: 1000 Primes

By Kevin Quiggle 

A recent issue of *"Midnite Software Gazette"* (issue 29, Nov.-Dec. 1985) published an article by Brian Junker in which run times were listed for 16 different languages calculating the first 1000 prime numbers. Unfortunately none of the 16 languages listed were COMAL. Luckily, our own *SPRITE* magazine (from Kevin's User Group) had already published a prime number program in Oct. 84. It was a simple matter to modify the program to calculate and time the generation of the first 1000 prime numbers. The results are shown in the table accompanying this article (I have borrowed all but the COMAL times from *"Midnite"*). Two times are shown for each language, one is for a run with the numbers calculated but not printed, the other is for a run with the numbers printed to the screen.

The results are impressive for COMAL. Not surprisingly, machine language gave the fastest time, with compiled "C Power" (which compiles to machine language) second. Coming in a very respectable third is COMAL, which was not only faster than BASIC, but was also faster than compiled BASIC. Even COMAL 0.14 was twice as fast as BASIC 7.0 running in **FAST mode**! This is not very surprising to those of us who have been using COMAL for some time. **The only thing that does surprise us is that some people are still using BASIC!**

[Ed note: we will try to get the program listings as used for this test. We do not have either the issue of Midnite or Sprite.]

[reprinted from SPRITE newsletter]

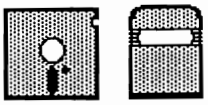
Rem	Print	No print	Language
a	32	26	Machine lang.
b	49	37	C Power
c	**95**	**57**	**COMAL 2.0**
d	**103**	**61**	**COMAL 2.0**
e	93	86	KYAN Pascal
f	100	93	Super Pascal
g	136	122	Oxford Pascal
h	**165**	**122**	**COMAL 0.14**
i	195	176	Speedwriter
j	198	178	Super C
k	235	---	Speedwriter
l	246	202	C64 FORTH
m	312	---	HES FORTH
n	328	304	BASIC 7.0 FAST
o	342	330	KYAN Pascal
p	427	---	SIMON'S BASIC
q	509	490	BASIC 2.0
r	696	634	BASIC 7.0
s	4503	4467	Commodore LOGO

(Ed note: in C128 fast mode, both COMAL times would be cut in half)

Remarks:

a. MOS Technologies
b. Pro-line, compiled to M.L.
c. Optimized using integer counters
d. No optimization
e. Kyan Advanced, compiled to M.L.
f. Abacus, compiled to p-code
g. Precision Software compiled to p-code
h. Integer flag array to conserve memory
i. CodeWriter; BASIC compiler used with "speed-up" options
j. Abacus, compiled to p-code (?)
k. BASIC compiled, no special options
l. Abacus (interpreted language)
m. HES (interpreted language)
n. C128 FAST mode (2 MHz) BASIC
o. KYAN Software (compiled to p-code)
p. Some Simon's BASIC enhancement were used (e.g.,WHILE-WEND)
q. Commodore-64 BASIC
r. BASIC 7.0 at 1 MHz on the C128
s. Estimated time: LOGO ran out of memory at the 306th prime. ■

Several Benchmarks



by Herbert Denaci

Here are the results of some benchmark tests. For background information, it is noted that I am a retired engineer with an interest in running Flight Dynamics simulations. That's the reason I wanted to compare the performance of various languages on the Commodore computer. The comparisons included BASIC 2.0 and 7.0, COMAL 0.14 and 2.0, Oxford Pascal, Promal, and Nevada Fortran (from Commodore).

The benchmark tests included:

*Ahl's Simple Benchmark: from *Creative Computing*, January 1984. *Today Disk #7* included a version which did not compute the Random function.

*Sieve of Eratosthenes: from *Commodore Microcomputers* May/June 1985 (with minor corrections). *Today Disk #5* included the corresponding COMAL and BASIC versions of the Sieve. This test was included because of the advertised performance of Promal.

*Trig Simple Benchmark: was generated to simply test the compute time for trigonometry functions typically used in Flight Dynamics problens.

*Flight Dynamics Simulation: is a representative guided missile program used for design studies. The COMAL game program "DOG/CAT" contains a simplified dynamics problem.

Runtimes for the various benchmarks and languages are shown below. COMAL 2.0 is fastest for Ahl's test, with Promal the slowest! The Sieve test results are similar to Promal's advertised performance. Commodore-128 BASIC 7.0 won the trig test in the "FAST" mode (operating at 2.0 MHz), however it is noted that C-128 BASIC is slower than C-64 BASIC for all tests in the normal mode of operation. It appears that except for Fortran, which uses the Z-80 chip in CP/M, all the languages use a similar routine for calculating the trig functions. The "Flight" benchmark comparisons include printout of the simulation results, and are my primary interest. **Here again, COMAL 2.0 has the shortest runtimes.** This was a surprise since I expected that "compiled" languages such as Pascal, Promal, and Fortran would always be faster than BASIC and COMAL. Also note that additional compile times is required for Pascal, Promal, and Fortran (on the C-128).

Of course, benchmark tests such as these are useful only if they are representative of the intended use of the computer. They also give little indication of the ease of programming. In comparing the five languages, there is no doubt that COMAL is superior. **It is user friendly!** BASIC isn't as difficult as the "compiled" languages, but it has the limitation of only recognizing the first two letters in variable names. Using compilers makes it difficult and time consuming to trouble shoot and edit a program. The line editor in FORTRAN with CP/M is particularly difficult.

COMPILE TIMES in seconds

Ahl	Trig	Flight	Sieve	Language
12	7	94	12	Pascal
53	51	91	12	Promal
89	66	230	---	Fortran
0	0	0	0	**COMAL**
0	0	0	0	BASIC

More ►

Several Benchmarks - continued

RUNTIME in seconds - Ahl's

25.5 COMAL 2.0
63.0 Oxford Pascal
63.6 C128 BASIC fast mode
95.0 Nevada Fortran
111.0 COMAL 0.14
114.8 C64 BASIC
166.0 Promal

RUNTIME in seconds - Trig

33.0 C128 BASIC fast mode
58.2 COMAL 2.0
59.5 COMAL 0.14
59.5 C64 BASIC
62.5 Oxford Pascal
67.5 Promal
172.0 Nevada Fortran

RUNTIME in seconds - Flight

201 COMAL 2.0
242 Oxford Pascal
245 C128 BASIC fast mode
251 COMAL 0.14
300 Promal
337 C64 BASIC
468 Nevada Fortran

RUNTIME in seconds - Sieve

21.0 Promal
52.5 Oxford Pascal
114.5 COMAL 2.0
261.1 C128 BASIC fast mode
279.6 COMAL 0.14
378.9 C64 BASIC
----- Nevada Fortran

[Ed note: The time for both COMAL's would be cut in half on a C128 with fast mode enabled]

I hope that these results are of some use in spreading the word on COMAL.

[These programs are on Today Disk #12] ■

Quick Test

by Craig Van Degrift

I have done a simple benchmark comparison of various languages for the IBM PC. The program is:

```
FOR x#=1 TO 1000 DO
  y=SQR(SIN(x#/1000))
ENDFOR x#
```

The results on an IBM PC AT were:

0.77 IBM PC COMAL 2.0 (8087)
0.99 Borland Turbo Pascal 3.01a (8087)
1.1 Microsoft Fortran 77 3.20 (8087)
1.8 IBM BASIC Compiler 1.00
5.8 Borland Turbo Pascal 3.01a
7.2 IBM PC COMAL 2.0
10.8 IBM BASIC A 3.0

The results on a Columbia MPC 1600 were:

0.96 IBM PC COMAL 2.0 (8087)
1.10 Borland Turbo Pascal 3.01a (8087)
15.9 Borland Turbo Pascal 3.01a
24.5 IBM PC COMAL 2.0

BENCHMARK REFERENCES:

Sieve Benchmark, COMAL Today #5, page 1
Compute! Benchmark, COMAL Today #5, page 1
BASIC vs COMAL - Time Tests, COMAL Today #2, page 35
Benchmarking, Byte Jan 86, page 371
Programming May Make Sense For Business, PC Magazine Oct 29, 1985, page 108
Leaks Like A Sieve, Byte Aug 85, page 33
Benchmarks, Byte Aug 85, page 132
Creative Computing Benchmark, Creative Computing Jan 84, page 5
A High-Level Language Benchmark, Byte Sept 81, page 180
Some More On Performance Evaluation, Byte July 80, page 216 ■

Free Form Data Base

by Joel E. Rea

For those who were taken aback by my little April Fool's joke last issue, (*"GreyMAT - Think into your Computer"*) I have provided a genuinely useful little program. It is called *"FFDB"* for *"Free Form Data Base"*. It is designed not only as a simple to use Rolodex type program, but as a easy to follow example of structured COMAL programming, and will work, unchanged, with either version of C64 COMAL as well as IBM PC COMAL.

After displaying simple instructions, the program asks for your text input. The **last** character on each line you type is the command. Anything before the last character is considered data. The four commands are:

"." Adds the line to current file.

"?" Displays all lines in current file that contain the input text.

"^" Same as "?" above, but printer output.

"@" Exit the program (STOP key also works safely)!

Here is a sample transaction with FFDB (text user types is underlined, computer response is printed in italics, extra explanations inside parentheses()):

COMAL is faster than BASIC.
(text is added to file)

Len Lindsay is Captain COMAL.
(text is added to file)

faster than BASIC?
COMAL is faster than BASIC.

Captain COMAL?
Len Lindsay is Captain COMAL.

COMAL Today: (608) 222-4432.
(text is added to file)

Emergency: 911.
(text is added to file)

COMAL Today?
COMAL Today: (608) 222-4432.

COMAL?
COMAL is faster than BASIC.
Len Lindsay is Captain COMAL.
COMAL Today: (608) 222-4432.

COMAL^
(Above 3 lines print on printer)

?
This is FFDB.DAT.
COMAL is faster than BASIC.
Len Lindsay is Captain COMAL.
COMAL Today: (608) 222-4432.
Emergency: 911.

@
END AT 0310

Before running the FFDB program, you must first create a file called "FFDB.DAT" for it to work with. To create the file type in and RUN the following program (if you wish to start over, delete the old file first):

```
0010 open file 8,"ffdb.dat", write
0020 print file 8: "This is FFDB.DAT"
0030 close file 8
```

The FFDB program listed below shows how easy it is to understand a COMAL program. Even with no comment lines, anyone with any programming sense should have no trouble understanding it.

More ►

```
0010 print "free form database"
0020 print
0030 print "last character is command:"
0040 print "  "".."" to add to file,"
0050 print "  ""?"" to search file,"
0060 print "  ""^"" search to printer,"
0070 print "  ""@"" to exit program."
0080 print
0090 print "your input is normal."
0100 print chr$(18),
0110 print "my output is reversed."
0120 print
0130 dim line$ of 80, text$ of 80
0140 dim command$ of 1
0150 //
0160 repeat
0170   get'line'from'user
0180   case command$ of
0190   when "."
0200     add'line'to'file
0210   when "?"
0220     display'matches
0230   when "^"
0240     select output "lp:"
0250     display'matches
0260     select output "ds:"
0270   otherwise
0280     null
0290   endcase
0300 until command$="@"
0310 end
0320 //
0330 proc get'line'from'user
0340   repeat
0350     input "": line$
0360   until len(line$)>0
0370   lastchar:=len(line$)
0380   command$:=line$(lastchar:lastchar)
0390   if lastchar>1 then
0400     line$:=line$(1:lastchar-1)
0410   else
0420     line$:=""
0430   endif
0440 endproc get'line'from'user
0450 //
0460 proc add'line'to'file
0470   open file 8,"dat.ffdb",append
0480   print file 8: line$
0490   close file 8
0500 endproc add'line'to'file
0510 //
0520 proc display'matches
0530   open file 8,"dat.ffdb",read
0540   repeat
0550     input file 8: text$
0560     if line$ in text$ then
0570       print chr$(18),text$,"."
0580     endif
0590   until eof(8)
0600   close file 8
0610 endproc display'matches
```

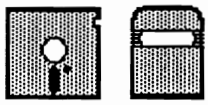
While RUNning, the FFDB accepts commands as lines of up to 80 (0.14) or 120 (2.0) characters. The LAST character specifies the command itself, while the rest of the line forms the "data". Thus, FFDB commands resemble English sentences!

As you can see, the operation of FFDB is so simple as to be almost intuitive! It makes a great Rolodex, except for the fact that you can neither sort nor edit using FFDB itself, but since the file is standard SEQ PETSCII, you can use most any editor or WP to do the job! I could have added those features, but I decided to keep it simple as a programming example. Consider these extensions as an *"exercise for the reader."*

Further Reference:

Electronic Phone, COMAL Today #10, page 45
Data Base Manager, COMAL Today #8, page 25
Mailing List Maker, COMAL Today #7, page 69
Data Base Manager in COMAL 0.14, COMAL Today #6, page 47
Distribution of a Disk Library List, COMAL Today #6, page 50 ■

Transfer 0.14 Programs To 2.0



by Captain COMAL

You decided to make the switch. You moved up to COMAL from BASIC a few months ago, and now you want to move up to the COMAL 2.0 Cartridge. But you don't want to lose all your COMAL 0.14 programs. Don't worry. COMAL is a standardized langauge. You should be able to use most of your programs. This article will explain how you can transfer programs from one version to another plus give you some tips on how to make the transfer easier.

First, keep in mind that each different version of COMAL has its own compression method - how it tokenizes and stores a COMAL program internally. They all may LIST a COMAL program that looks the same to you, but internally, each COMAL interpretter stores it in a unique manner. When you SAVE a COMAL program to disk, it is written in the compressed form. Therefore, it can only be LOADed again with the same version of COMAL. This presents no problems during normal use of COMAL. However, transferring programs is a special case. And, as you may have suspected, COMAL provides for this.

All COMAL systems have the capabilty to LIST a program to disk. This results in a standard ASCII file that is compatible with other versions and even other computers. Any program you wish to transfer to another version of COMAL must be LISTed to disk first. For example:

LIST "name"

Once your programs are LISTed to disk you are ready to begin the transfer. Just as all COMAL systems can LIST a program to disk as an ASCII file, they also can ENTER a program from an ASCII file. For example:

ENTER "name"

COMAL treats a program being ENTERed as if it were being typed in at the keyboard. Thus, it is analyzed, line by line as it is entered. If COMAL finds a problem with a line, it will temporarily pause entering the program and list the line on the screen with an error message. You then have the opportunity to correct the line - or simply insert a "!" after the line number to 'comment it out'. Once you hit RETURN and the line is accepted, COMAL resumes entering the program from disk.

Since COMAL is standardized, a COMAL program written for one version should work with another - if it sticks with the standard COMAL features. Of course, each version of COMAL will have its 'added' features. If you use those, your program is not guaranteed to be transportable. Generally speaking, most programs can be transported from COMAL 0.14 to COMAL 2.0 with little difficulty. But just in case, here are some tips, notes, and technicalities that might help:

***** FOR loop**

C64 COMAL 2.0 and IBM PC COMAL (both by UniCOMAL) consider the variable used in FOR loops as a LOCAL variable. Once the loop is finished, the variable is discarded. C64 COMAL 0.14 and PET COMAL 0.14 retain the last value of the loop variable. Other COMAL's may do other things. The COMAL standard defines the condition of the FOR variable after the loop as **undefined**. That means - don't

More ►

count on using a FOR loop variable after the loop is finished! This rarely will affect a properly written COMAL program.

***** Variable name / keyword conflict**

Another version of COMAL may include some added commands or keywords not present in your original COMAL version. There is a possiblility that a variable or procedure name in your program is a reserved keyword in the new COMAL version. Often a COMAL 0.14 program will include a procedure that imitates one of the added keywords of COMAL 2.0. For instance, PAGE is a keyword in COMAL 2.0 that will clear the text screen. A COMAL 0.14 program could 'add' that word to its program like this:

```
PROC page
  PRINT CHR$(147),
ENDPROC page
```

In this case, all the places in the program where PAGE is found will work unchanged. But the procedure that defines PAGE must be deleted from the program. While ENTERing the program, just insert a ! right after the line number to comment out these lines. Once the entire program is entered, you can go back and delete the procedure. This same process applies to the use of PI as a variable name. PI is predefined in COMAL 2.0, so if your COMAL 0.14 defines PI in your program (and it is the same kind of PI) simply comment out the definition line and delete it later.

It's also possible that your use of a new reserved keyword is different. In this case, a quick solution is to add an apostrophe (') to end of your variable name. For example, if you used a variable named PAGE to count how many pages have been printed, just change it to PAGE' since the ' is a valid character and can be used as part of a variable name.

***** CLOSED PROCedures and FUNCtions**

In COMAL 0.14 a CLOSED procedure has LOCAL variables - but other procedure and function names in the program are all GLOBAL. In COMAL 2.0 both variables AND procedures and functions are LOCAL. An IMPORT statement must be added for any procedure or function used within a CLOSED procedure. For example, the procedure BOLD'CHAR is called from within the following CLOSED procedure:

```
PROC bold(text$) CLOSED
  IMPORT bold'char
  FOR x=1 TO LEN(text$) DO
    bold'char(text$(x)
  ENDFOR x
ENDPROC bold
```

In COMAL 0.14 the procedure would not have included the IMPORT statement, since all procedures are treated as GLOBAL and are automatically imported.

One other technical note on this subject. The USE command activates a package which in turn may include procedures and functions. These are also 'locked out' of CLOSED procedures. However, you have a choice of how to deal with this. Either use an IMPORT statement for the ones needed, or just reissue the USE command inside the CLOSED procedure! Examples:

```
USE graphics
// program here
PROC box(side) CLOSED
  IMPORT forward, left
  FOR x=1 TO 4 DO
```

More ►

```
    forward(side)
    left(90)
  ENDFOR x
ENDPROC box
//
PROC square(side) CLOSED
  USE graphics
  FOR x=1 TO 4 DO
    forward(side)
    left(90)
  ENDFOR x
ENDPROC square
```

Both BOX and SQUARE use the added commands in the GRAPHICS package. BOX used IMPORT to access the commands needed. SQUARE used another USE statement.

***** GRAPHICS and SPRITES**

In COMAL 0.14 there are graphics and sprite commands built in. COMAL 2.0 includes those same commands plus many more - but not as part of COMAL. They are in packages! To access them you need to add a USE statement at the beginning of your program. Also, beware of the CLOSED procedure situation (see above). Finally, all parameters used by graphics and sprite commands require parentheses in 2.0. For example:

FORWARD 50 *// <--- COMAL 0.14*
FORWARD(50)*// <--- COMAL 2.0*

Graphics and sprites are not considered part of COMAL and are not covered in the COMAL standard. However, going from COMAL 0.14 to COMAL 2.0 has no problem, since they both were written by UniCOMAL and COMAL 2.0 is upward compatible. But manually adding all those parentheses can be a pain.

Fear not. COMAL to the rescue!

Since you now have the COMAL 2.0 Cartridge, why not make use of one of its features: user defined function keys! You can set up your function keys so that they will do most of the work for you. ENTERing even large graphics programs then becomes a matter of tapping the function keys once per line as needed. Before you start ENTERing your programs, set up your function keys with the following commands:

```
USE system
defkey(5,"("13")"13"")
defkey(7,"("13""157")"13"")
```

Now, as the program is ENTERed, COMAL will stop and display lines with missing parentheses. If the line includes a comment use the *F7* key. Otherwise *F5* will add the parentheses.

Also, note that the command CLEAR in COMAL 0.14 is now CLEARSCREEN in 2.0, SETTEXT is now TEXTSCREEN, and SETGRAPHIC is now GRAPHICSCREEN.

One other difference is that the turtle's HOME position is always at coordinate 0,0 in COMAL 2.0. After issuing a USE GRAPHICS command, this is the bottom left corner of the screen, rather than the center of the screen as in COMAL 0.14. To 'fix' this, just redefine the HOME command in your program by adding this procedure:

```
PROC home
  moveto(160,99)
ENDPROC home
```

***** BACKGROUND, BORDER, PENCOLOR**

COMAL 2.0 gives you the ability to use different colors on your text screen

More ►

than on your graphics screen. This is a very nice feature. However, in order to do this, they had to add 3 new commands. BACKGROUND, BORDER, and PENCOLOR still operate the same on the graphics screen. However, to change colors on the text screen use these new commands:

USE GRAPHICS
TEXTCOLOR(n) *// pencolor*
TEXTBACKBROUND(n) *// background*
TEXTBORDER(n) *// border*

These new commands are part of the graphics package, thus you need a USE GRAPHICS before you can access them, even if you are not doing any graphics work. USE TURTLE also includes them.

One other new way to control the colors on your text screen is with the command TEXTCOLORS defined in the SYSTEM package:

USE SYSTEM
TEXTCOLORS(12,0,1)

The order of the three colors is: border, background, textcolor. TEXTCOLORS can be used to set all three colors at once, or just one. For any color you do not want to change just use -1 in its place.

***** Sprites**

Just as with the graphics commands, sprite commands were part of COMAL 0.14, but in COMAL 2.0 are part of a package. Parentheses are also now required in COMAL 2.0 but were not used in COMAL 0.14. Use the same function key definition explained above to add the parentheses as you ENTER the program. Remember to add a USE SPRITES command at the start of the program. Finally, the IDENTIFY command automatically turned on a sprite in COMAL 0.14. So many users disliked this that SHOWSPRITE was made a separate command in COMAL 2.0. So, whenever a COMAL 0.14 program uses IDENTIFY, you may need to add a SHOWSPRITE command after it.

***** Arrays as Parameters**

Entire arrays can be passed to a procedure or function as a parameter. To do so in COMAL 0.14 you simply used the array name. In COMAL 2.0 you have to use the array name and a set of empty parentheses (with an imbedded comma for each added dimension):

<u>COMAL 0.14:</u>

```
DIM PLAYER$(1:3) OF 9, TABLE(1:9,1:3)
...
SHOW'IT(PLAYER$,TABLE)
...
PROC SHOW'IT(REF PLAYER$,TABLE)
```

<u>COMAL 2.0:</u>

```
DIM PLAYER$(1:3) OF 9, TABLE(1:9,1:3)
...
SHOW'IT(PLAYER$(),TABLE(,))
...
PROC SHOW'IT(REF PLAYER$(),TABLE(,))
```

***** Keyboard Buffer**

In COMAL 0.14 you could access the keyboard buffer (key count at 198, buffer at 631-640) from within a program. In COMAL 2.0 this not always works. Very few programs did this. Those that did may need some changes to work in COMAL 2.0.

More ►

***** UNIT 9 Specification**

In COMAL 0.14 the keyword UNIT was used to specify a special unit device number, primarily for UNIT 9 disk drive access. COMAL 2.0 uses the drive number to specify which drive is to be used, counting up from "0:":

"0:" Unit 8 Drive 0
"1:" Unit 8 Drive 1
"2:" Unit 9 Drive 0
"3:" Unit 9 Drive 1
etc.

```
open file 2,"0:name",unit 9,read//0.14
open file 2,"2:name",read       //2.0
```

Secondary address was also accessed via the UNIT keyword in COMAL 0.14. In COMAL 2.0 that information is included as attributes at the end of the file name:

```
"lp:/s7" //<---sec address of 7
```

One other major use of UNIT in COMAL 0.14 was to override the default printer open used by SELECT. Most users wanted their printer in lower case mode while COMAL automatically opened it in uppercase / graphic chracters mode. Change the lines:

0.14:

```
open file 255,"",unit 4,7,write
select "lp:"
```

2.0:

```
select "lp:"
```

Also, if CLOSE FILE 255 was included in the COMAL 0.14 program after the printing was done, it may be discarded in the 2.0 program. SELECT "DS:" will do the trick in both versions.

***** FRAME**

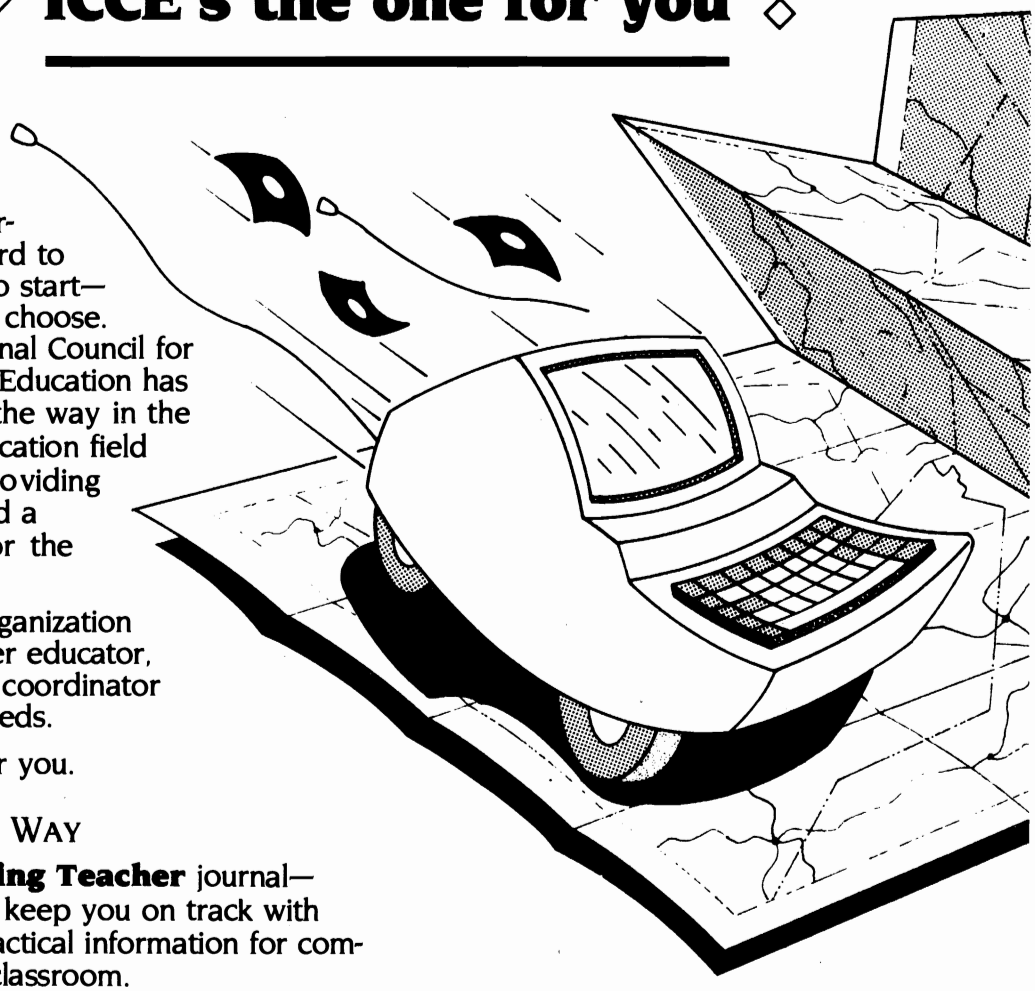
FRAME is a very rarely used keyword in COMAL 0.14. In COMAL 2.0 it is changed to WINDOW as part of the graphics package.

REFERENCES

COMAL 2.0 Keywords, COMAL Today #11, page 30
Connect Your C64 To An IBM PC, COMAL TODAY #9, page 12
Easy Sprites For Beginners, COMAL TODAY #8, page 12
COMAL Standards Conference, COMAL TODAY #7, page 8
Graphics Kernal (proposal), COMAL TODAY #7, page 8
Getting Started With COMAL 2.0, COMAL TODAY #6, page 3
COMAL Cartridge Programming Tips, COMAL TODAY #6, page 6
Moving Up To COMAL 2.0, COMAL TODAY #6, page 8
COMAL 2.0 From 0.14, COMAL TODAY #6, page 12
Compare Commands: 2.0 & 0.14, COMAL TODAY #6, page 13
COMAL 2.0 Drive Numbers, COMAL TODAY #6, page 36
PASS Command For Drive 9, COMAL TODAY #6, page 37
Passing An Array As A Parameter, COMAL TODAY #6, page 39
COMAL Is Compatible, COMAL TODAY #6, page 53
Define Function Keys Batch File, COMAL TODAY #6, page 62
COMAL 2.0 Is Compatible With 0.14, COMAL TODAY #6, page 79
COMAL 0.14 Machine Code Routines, COMAL TODAY #5, page 26
Pass An Array As A Parameter, COMAL TODAY #4, page 48 ■

Take Off With Us

◇ ICCE's the one for you ◇



With today's profusion of computer information, it's hard to know where to start—which road to choose. The International Council for Computers in Education has been guiding the way in the computer education field since 1979, providing leadership and a ground plan for the future.

It's the one organization every computer educator, administrator, coordinator or librarian needs.

It's the one for you.

GUIDING THE WAY

The Computing Teacher journal—guaranteed to keep you on track with up-to-date, practical information for computers in the classroom.

Special Interest Groups—share information to help your special interest area grow. SIGs include computer coordinators, teacher educators, administrators and special educators, and are planned for advanced placement in computer science, community colleges and videodisc users. The quarterly **SIG Bulletin** serves as a forum for SIG information.

Booklets and Monographs point to additional information on specific topics. **ICCE Packets** provide you with teacher training materials. Members receive a 10% discount on all three.

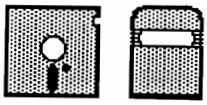
ICCE Committees address a variety of ethical and practical issues important to you as a computer-using educator.

ICCE participates in computer education conferences throughout the world, supporting our state- and region-wide member organizations.

INTERNATIONAL COUNCIL FOR COMPUTERS IN EDUCATION — ICCE

Join the One for You.

COMAL Monitors

by Joel Ellis Rea 

My COMAL Machine Language Monitors (CMLMs) are not true MLMs in the usual sense. These are actually collections of PROCedures and FUNCtions that give the user many of the functions of a MLM from within the COMAL Editor. These include Hex/ASCII Dumps, ASCII Display and Disassembly, as well as Decimal-to-Hex and Hex-to-Decimal, plus the ability to PEEK and POKE words and integers, and PEEK strings. Sorry, no mini-assembler and no standard MLM style commands. You use the normal COMAL environment. There is a monitor for both versions of COMAL on *Today Disk #12*. Both have the same features. Start this monitor like this:

chain "monitor"

The monitor reads the SEQuential file *"dat.6510opcodes"*, which contains the opcode info for the disassembler PROCedures. You then drop back into the COMAL editor, and can use any of the following PROCedures and FUNCtions, as well as any COMAL commands:

FUNC DEC(hex$) -- converts a hexadecimal number contained in hex$ to a decimal number (0.14 monitor only - it is built into COMAL 2.0 as the $ hex constant).

PROC HPRINT1(dec#) -- prints a single hexadecimal digit, converted from dec# (must be in range 0:15).

PROC HPRINT2(dec#) -- prints 2 hexadecimal digits, converted from dec# (in range 0:255).

PROC HPRINT4(dec) -- prints 4 hexadecimal digits, converted from dec (in range 0:65535).

FUNC WPEEK(addr) -- returns value of 2 byte word address at locations addr and addr+1, with addr+1 as the MSB.

PROC WPOKE(addr,val) -- places val in locations addr (LSB) and addr+1 (MSB).

FUNC DPEEK(addr) -- returns value of double-byte integer at locations addr (MSB) and addr+1 (LSB).

PROC DPOKE(addr,val) -- places val in locations addr (MSB) and addr+1 (LSB).

PROC TYPE(filename$) -- displays non RELative file filename$ to the screen, or to currently SELECTed OUTPUT file. C2.MLM version is especially fast!

PROC DUMP1LINE(addr) -- Displays one line on the screen, consisting of the hexadecimal values of the 8 bytes starting at addr (rounded to lower 8), then the ASCII character equivilants.

PROC DUMP(lines#,addr) -- DUMPs lines# lines of 8 bytes each starting at addr (rounded to lower 8).

PROC DUMP'RANGE(start,end) -- DUMPs enough lines to cover the range start:end.

PROC DISP1LINE(addr) -- Displays one line on the screen, consisting of the ASCII values of the 32 bytes starting at addr (rounded to lower 32).

PROC DISP(lines#,addr) -- This is to DISP1LINE what DUMP is to DUMP1LINE.

PROC DISP'RANGE(start,end) -- Is to DISP1LINE what DUMP'RANGE is to DUMP1LINE.

More ►

PROC DISASM1LINE(addr) -- Displays one line on the screen, consisting of the disassembled version of the 6510 ML instruction starting at addW. May be 1 to 3 bytes.

PROC DISASM(lines#,addr) -- This is to DISASM1LINE what DUMP is to DUMP1LINE. DISASM and DISASM'RANGE both display instructions that might be 'hidden' under BIT instructions inside angle brackets <>.

PROC DISASM'RANGE(start,end) -- This is to DISASM1LINE what DUMP'RANGE is to DUMP1LINE.

NOTE: Recently, I came across an article in The Transactor, Nov. 85, page 50, which describes some undocumented 6502/6510 opcodes. The author, Jim McLaughlin of Ottowa, Ontario, stated that at this point, there are no disassemblers that can handle them. Well, there is now! My CMLMs recognize the undocumented opcodes as described in Jim's article. They treat "SKB" and "SKW" opcodes, along with the only "???" opcode left ($BB), the same as "BIT", i.e. since these opcodes can be used to "hide" other instructions, any instructions "hidden" inside them are shown on the following lines, surrounded by angle brackets <>! Output displays can be directed to the printer with the SELECT command.

Further Reference:

Machine Langauge Monitor Package, COMAL Today #10, page 74
Disassembler In COMAL 2.0, COMAL Today #7, page 59
SMON-COMAL, monitor program on COMAL 2.0 Packages disk. ■

Letters

COMAL IN SCHOOL

Dear Mr. Lindsay- I teach computer programming at the local school, Pocahontas County High School. This coming term will be the third year of the offering. Thanks to an administration which is looking to provide proper training for their students, I was able to convince them to do away with BASIC and offer COMAL in its place.

The past term, for one class of second year programmers, I did introduce COMAL for two thirds of the year and Pascal for the balance. I based my materials on Atherton's book, *Structured Programming With COMAL.*

Unfortunately, American education is now buried in the **process** of educating its youth, and not in educating (I've spent 34 years in the classroom earning the right to that bias). **BEHAVIORAL OBJECTIVES!!** is the hue and cry of the day (heck with the student is the unrecognized message).

Therefore, I have been putting together a text for my classes, complete with objectives, written for students with the background I will get in my classes (many will not even have had a class in Algebra).

COMAL is quite a fantastic language. May you and COMAL keep up the good work; together, we can find a deserved lock in the American schools. - J William Leary, Dunmore, WV

His 260 page spiral bound book should be available by the time you read this. It is Introduction to Computer Programming With COMAL 2.0. A separate 64 page answer book is also available. See the order form at the back of this issue. ■

Fast DIR Revisited

by Ray Carter 

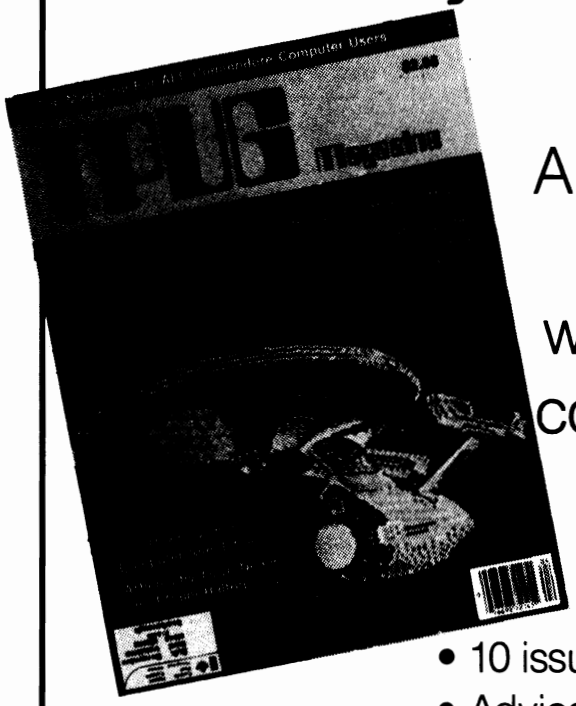
COMAL Today #7 and #8 have printed procedures for reading a directory into memory fast. This method is fine, except it restricts reading directories to 4040 type disk drives.

The problem is that each of these procedures assume the directory is on track 18, which is true for the 1541, but not for a Commodore 8050 or 8250. If we read the directory in as a sequential file, the speed can still be maintained and make the directory reader will be compatible with any disk drive that connects to a Commodore computer. An example is listed below:


```
DIM d$ OF 16,id$ OF 2,n$(144) OF 16,t$(144) OF 4,b#(144)
read'dir(d$,id$,n$(),t$(),c,b#())
PAGE
PRINT "DISK:";d$;" ID: [",id$,"]"
PRINT "Type Blk Name"
PRINT "==== === ================"
FOR x:=1 TO c DO
 PRINT t$(x),TAB(6),
 PRINT USING "###": b#(x);
 PRINT n$(x)
ENDFOR x
//
PROC read'dir(REF d$,REF id$,REF name$(),REF t$()
,REF num,REF b#()) CLOSED // wrap line
 DIM block$ OF 32,junk$ OF 1
 num:=0
 TRAP
  MOUNT
  OPEN FILE 78,"u8:$0/s1/t+/d+",READ
  junk$:=GET$(78,142)
  block$:=GET$(78,27)
  d$:=strip$(block$(1:16))
  id$:=strip$(block$(19:20))
  junk$:=GET$(78,85)
  REPEAT
   block$:=GET$(78,30)
   IF ORD(block$(4))>0 THEN
    num:+1
    CASE ORD(block$(1)) MOD 16 OF
    WHEN 0
     t$(num):="del"
    WHEN 1
     t$(num):="seq"
    WHEN 2
     t$(num):="prg"
    WHEN 3
     t$(num):="rel"
    WHEN 4
     t$(num):="usr"
    OTHERWISE
     t$(num):="***"
    ENDCASE
    IF ORD(block$(1)) bitand 64 THEN t$(num):+"<"
    name$(num):=strip$(block$(4:19))
    b#(num):=ORD(block$(29))
    b#(num):+ORD(block$(30))*256
   ENDIF
   IF (num MOD 8)<>0 THEN junk$:=GET$(78,2)
  UNTIL EOF(78)
 HANDLER
 ENDTRAP
 CLOSE FILE 78
 //
 FUNC strip$(string$)
  IF CHR$(160) IN string$ THEN
   RETURN string$(1:(CHR$(160) IN string$)-1)
  ELSE
   RETURN string$
  ENDIF
 ENDFUNC strip$
ENDPROC read'dir ■
```

TPUG

MORE THAN JUST ANOTHER COMPUTER MAGAZINE!



A membership in the world's largest computer club will provide you with:

- 10 issues of TPUG magazine
- Advice from experts like Jim Butterfield and Elizabeth Deal
- Access to our huge library of public domain software
- Access to our online help services
- All for only $25/yr.

TPUG
101 Duncan Mill
Suite G7
Don Mills, Ontario
CANADA
M3B 1Z3

JOIN US NOW!!

I want to join TPUG as an associate member.

Name _____ Home phone (____) _____

Address _____ Work phone (____) _____

City/Town _____ ☐ Cheque enclosed

Prov/State _____ ☐ Money order enclosed

Postal/Zip Code _____ Country _____ ☐ Visa ☐ MasterCard

My computer equipment is: _____ Card# _____

_____ Expiry _____

_____ Signature _____

Autoexec

by Scott Strool 

One of the more powerful capabilities of your COMAL 2.0 Cartridge is the ability to set up batch files to execute any number of commands, this is probably no secret. Also being a user of an IBM PC, I enjoyed having the same power on my 64. Alas something was missing. I want my batch files to execute when I power my computer. With a little trickery and COMAL magic you can simulate an automatic executing batch file.

The normal way to have a batch file execute would be to type the command:

SELECT INPUT "batchfile name"

Not very automatic. With a few simple steps you will be able to almost automatically start up any batch file when your 64 is first turned on.

First create a batch file that you would like executed as soon as your computer is turned on. This could define your function keys, change screen colors, run a menu program or any set of commands that you would type by hand. If you already have a good batch file then skip this step. Second, create a one line COMAL program like this one:

```
0010  SELECT INPUT "bat.autoexec"
```

"bat.autoexec" is the name of the batch file you want auto started. Save this one line program with the name *"autoexec"*. Now the COMAL magic comes in. Look at the directory of one of your programming work disks. The very first file is the important one. Copy that file on the same disk (with a different name of course):

```
COPY "firstfile","firstfile+"
```

Now DELETE the first file in the directory, and rename the copied file back to the original file name:

```
DELETE "firstfile"
RENAME "firstfile+","firstfile"
```

Now, re-SAVE the autoexec program on this disk and it should become the first program in the directory:

```
SAVE "autoexec"
```

Check to see that it is the first file in the disk directory and make sure your *"bat.autoexec"* file is also on the disk. It can be located anywhere in the directory.

Finally, test the system as if you were just starting a programming session. Insert the disk and press SHIFT Run/Stop key. This will RUN the first program in the disk directory: your *autoexec* program. The program will then execute your batch file. Not completely automatic but one keystroke is close.

Further references:

COMALites Unite, Autoboot EPROM, COMAL Today #12, page 2
Function Keys For MSD Dual, COMAL Today #9, page 32
Function Keys, COMAL Today #9, page 33
More Function Keys, COMAL Today #9, page 33
Function Keys Revisited, COMAL Today #9, page 33
Programming Batch Files - 2.0, COMAL Today #8, page 50
Batch File Use, COMAL Today #7, page 30
Making Batch Files, COMAL Today #7, page 58
COMAL Batch Files, COMAL Today #6, page 52

Now Available Through Aquarian Software

Gold Disk Series

Volumes 1 through 11 Now Available!!!

Volume 11 Features a C-64 Assembler

Each Disk Contains:

- The Monthly Feature Program
- Programming Tutorials
- High Quality Games
- And Much More

Gold Disk Series for 128
Coming Soon!

Only $14.95 Per Disk*

* Plus Shipping and Handling

The Cataloger

The Ultimate Disk Cataloging System for the 64!

Features of The Cataloger V3.5A include:

- ★ Loads directly from the disk itself.
- ★ Ability to change name of entry.
- ★ Fast — Uses relative files exclusively
- ★ Search, Sort and Print by any of 12 fields.
- ★ 1100-program (or disk) capacity per data disk.
- ★ All machine language.
- ★ Menu driven — very easy to use.
- ★ Works with one or two drives.

Only $24.95

BobsTerm Pro

The Ultimate Terminal Software !

Upload / Download Supports Punter, X-Modem, XON / XOFF, DC1 / DC2, and Much More!

28.5 Byte Buffer with unmatched editing abilities

- User Adjustable Parameters
- 10 Custom Character Sets
- Unlimited Phone Book Storage
- Programmable Macro Command Strings

Only $59.95

MATRIX — NOW AVAILABLE!!

The Indispensable C-128 Utility / Starter Kit !

Use **dozens** of 128 features in the 64 mode:

- Numeric Key Pad
- Cursor Keys
- 80-Column RGB Output
- Many Other Special Function keys

One-Key Functions Include:

- **2 Megahertz "Fast Mode"**
- **One-Key Screen Dumps**
- **Full-Featured DOS Utility Menu**

Other Features Include:

- Fast Loading
- Fast Copy For The 1571!
- Relocatable In Memory
- 100% Transparent to BASIC

Available Now For Only $59.95

Graphic Screen Exporter

A Universal Graphics Converter !

Converts Anything to Anything — Including:

Koala Pad	Doodle
Flexidraw	Print Shop
COMAL	Paint Magic
CAD GEM	Micron Eye

And Many Many More ! !

The Most Versatile Graphics Utility Ever Released for the Commodore 64 !

Only $29.95

ALSO AVAILABLE:

OmiTerm.....$19.95
Full-Feature Terminal at an Affordable Price!
Turbo Calc/64.....$17.95
A great spreadsheet at an Unbelievable Price!
Tax Computation.....$29.95
The friendliest tax package on the market.
Guitar Master.....$49.95
A comprehensive musical instruction package
Fast Boot!.....$14.95
Mike J. Henry's Fast Loader for 1541/MSD
Thriller Collection.....$24.95
Seven intricate text adventures on one disk

Call or Write for Full Catalog !

CAD-GEM

Computer Assisted Design
Graphic Element Manipulation

A Wire Frame CAD system for the C64 !
Input from Joystick, Track Ball, Light Pen or Graphics Tablet
360 Degree Rotation in .1 Degree Increments
Scaling on a 64K x 64K, 2048 Mega-Bit Virtual Screen
Independent Manipulation of 400 Objects (Points or Lines)

You must see CAD GEM to believe it!
Demo Disk Available for $3.00

$89.95

MODEM MASTER

The Friendliest Commodore BBS Available

- Works with 1541 or MSD Dual Drive
- 300 / 1200 Baud Operation
- New Punter File Transfer Protocol
- Sub-Directories for File Transfer
- 250 User Capacity
- Accurate Clock / Calendar
- Printer Output
- Information Files
- "Old" E-Mail Deleted After One Week
- Set Up In Only 10 Minutes !

Only $29.95

Total Software Development System

by Kevin Pickell
Now Available in the States !

Assembler/Editor — fast load, get, log and loadat; adds 38 new commands; full macro instructions; allows 13-character labels; assembles to and from disk
Sprite Editor — 256 sprites in memory, view 64 at same time, works with keyboard, joystick or trackball, animates sprites during design
Unassembler — create source code from any ML program
Sound Editor — create interrupt-driven sound effects
Character Editor — edit all characters. Screens to 255x64. Hi-res & Multi-color Character Sets

TSDS automatically includes sprites, characters, mattes and sound effects into source code!

Only $39.95

128 Version Coming Soon !

Aquarian Software
P.O. Box 22184
Portland, OR 97222

VISA

To order, Call: (503) 654-2641
VISA & MasterCard Accepted

MasterCard

Add 3.00 S & H Per Order
(Add Additional $2.00 for COD)
Canadian Orders Add 10.00 S&H
Allow 3-4 Weeks For Delivery

Write or Call for Full Catalog — Dealer Inquiries Welcome !

Package Keywords

compiled by Daniel W Parish

These are the added commands in the built in COMAL 2.0 cartridge packages.

SYSTEM

initialize with: **USE SYSTEM**

BELL-ring bell specified number of times
bell(<number>)
bell(3)

CURCOL--returns cursor column position
curcol
c:=curcol

CURROW--returns cursor row position
currow
r:=currow

DEFKEY -- define function keys
defkey(<func key #>,<issue string$>)
defkey(1,"cat"+CHR$(13))

FREE --returns amount of free memory
free
mem'left:=free
PRINT free

GETSCREEN -- save textscreen
getscreen(<string$>)
getscreen(oldscreen$)

GETTIME$ -- returns current time
gettime$
this'time$:=gettime$

HARDCOPY -- text screen dump
hardcopy(<filename$>)
hardcopy("lp:/a+")

INKEY$ -- wait for key-press
inkey$
choice$:=inkey$
PRINT inkey$

KEYWORDS'IN'UPPER'CASE --
keywords'in'upper'case(<true/false>)
keywords'in'upper'case(TRUE)

NAMES'IN'UPPER'CASE --
names'in'upper'case(<true/false>)
names'in'upper'case(TRUE)

QUOTE'MODE -- quote mode on/off
quote'mode(<true/false>)
quote'mode(FALSE)

SERIAL -- toggle between serial/IEEE
serial(<true/false>)
serial(TRUE)

SETPAGE -- set current memory page
setpage(<page number>)
setpage(2)

SETPRINTER-define printer parameters
setprinter(<printer specs string>)
setprinter("u4:/a+/l+/t+/s7/d-")

SETRECORDDELAY -- random file
setrecorddelay(<amount delay>)
setrecorddelay(99)

SETSCREEN--restore saved textscreen
setscreen(<string$>)
setscreen(oldscreen$)

SETTIME -- set the real time clock
settime(<time string$>)
settime(<hh:mm:ss.t>)
settime("11:23:00.0")

SHOWKEYS--display function key defs
showkeys
showkeys

TEXTCOLORS -- set all colors at once
textcolors(<border>,<background>,<cursor>)
textcolors(6,6,1)

More ►

SOUND
initialize with: **USE SOUND**

ADSR -- set sound envelope parameters
adsr(<voice#>,<atk>,<decay>,<sust>,<rel>)
adsr(2,9,5,4,1)

ENV3 -- returns value of envelope 3
env3
wave'point:=env3

FILTER -- specify voices to filter
filter(<voice1>,<voice2>,<voice3>,<external>)
filter(TRUE,TRUE,FALSE,FALSE)

FILTERFREQ -- set filter frequency
filterfreq(<frequency 0-2047>)
filterfreq(1035)

FILTERTYPE -- set filtertype
filtertype(<low>,<band>,<high>,<vce3 off>)
filtertype(TRUE,FALSE,TRUE,TRUE)

FREQUENCY-returns frequency of note
frequency(<note$>)
this'freq=frequency("a3")

GATE -- control playing of notes
gate(<voice #>,<on/off>)
gate(2,FALSE)

NOTE -- set voice to note$
note(<voice #>,<note$>)
note(2,"c4#")

OSC3 -- returns value of oscillator 3
osc3
wave'value:=osc3

PULSE -- set pulse width
pulse(<voice #>,<width 0-4095>)
pulse(2,1295)

RESONANCE -- control resonance
resonance(<value 0-15>)
resonance(9)

RINGMOD -- control ring modulation
ringmod(<voice #>,<on/off>)
ringmod(2,TRUE)

SETFREQUENCY-set voice frequency
setfrequency(<voice #>,<freq val>)
setfrequency(2,1035)

SOUNDTYPE -- set voice waveform
soundtype(<voice #>,<waveform>)
soundtype(2,1)

waveform -- meaning
0 - no sound
1 - triangle
2 - sawtooth
3 - pulse (needs pulse too)
4 - noise

SYNC -- synchronize voice frequencies
sync(<voice #>,<on/off>)
sync(2,FALSE)

VOLUME -- set master volume
volume(<level>)
volume(15)

AUTOMATIC SOUND CONTROL
Program runs simultaneously with music.

PLAYSCORE --starts voices indicated
playscore(<voice1>,<voice2>,<voice3>)
playscore(TRUE,TRUE,TRUE)

SETSCORE -- set up music score
setscore(<voice#>,<freq array>,
<gate=1 array>,<gate=0 array>)
setscore(2,freq#(),on#(),off#())

STOPPLAY -- stop given voice
stopplay(<voice1>,<voice2>,<voice3>)
stopplay(FALSE,FALSE,TRUE)

WAITSCORE-returns TRUE if playing
waitscore(<voice1>,<voice2>,<voice3>)
IF waitscore(TRUE,TRUE,TRUE) THEN

More ►

SPRITES

initialize with: **USE SPRITES**

DATACOLLISION-collision with data?
datacollision(<sprite#>,<reset collsn?>)
IF datacollision(1,TRUE) THEN

DEFINE -- set up sprite image
define(<shape #>,<64 byte def$>)
define(1,sprite$)

HIDESPRITE -- turn off specified sprite
hidesprite(<sprite #>)
hidesprite(2)

IDENTIFY -- assign a shape to a sprite
identify(<sprite #>,<shape #>)
identify(2,9)

LINKSHAPE -- links sprite to program
linkshape(<shape #>)
linkshape(9)

LOADSHAPE -- load sprite definition
loadshape(<shape #>,<filename>)
loadshape(9,"shap.bat1")

PRIORITY -data priority over sprite?
priority(<sprite #>,<data priority?>)
priority(2,TRUE)

SAVESHAPE -- save sprite definition
saveshape(<shape #>,<filename>)
saveshape(2,"shap.mine")

SHOWSPRITE -- turn on sprite image
showsprite(<sprite #>)
showsprite(2)

SPRITEBACK -set 2 sprite multicolors
sprite back(<color#1>,<color#2>)
sprite back(2,6)

SPRITECOLLISION -sprite collision?
spritecollision(<sprite#>,<rest colsn?>)
hit=spritecollision(2,FALSE)

SPRITECOLOR -- set color of sprite
spritecolor(<sprite #>,<color #>)
spritecolor(2,9)

SPRITEINQ -- returns sprite data
spriteinq(<sprite #>,<item #>)
image=spriteinq(2,8)

item# function
---
0 - is sprite visible?
1 - color number of multi-color#1
2 - color number of sprite
3 - color number of multi-color#2
4 - expanded in width?
5 - expanded in height?
6 - multi-color or hi-res sprite?
7 - data priority
8 - image number
9 - is sprite moving?
10- sprite to sprite collision?
11- sprite to data collision?

SPRITEPOS -put sprite at x,y location
spritepos(<sprite#>,<x coord>,<y coor>)
spritepos(2,100,55)

SPRITESIZE -sprite size (expand or not)
spritesize(<sprite#>,<x exp?>,<y exp?>)
spritesize(2,TRUE,FALSE)

SPRITEX -- returns x coord of sprite
spritex(<sprite #>)
x'pos:=spritex(2)

SPRITEY -- returns Y coord of sprite
spritey(<sprite #>)
y'pos:=spritey(2)

STAMPSPRITE-copy sprite to screen
stampsprite(<sprite#>)
stampsprite(1)

More ►

AUTOMATIC SPRITE CONTROL

initialize with: **USE SPRITES**

Sprites are moved by the COMAL system. Program continues running simultaneously.

ANIMATE -- auto sprite movement
animate(<sprite number>,<sequence$>)
animate(2,sequence$)
sequence$ made from the following:
(each is a set of two bytes)

SHAPE & DURATION - set time for shape
CHR$(<shape #>)+CHR$(<duration>)
""2""20""

SPRITE COLOR - color to use
"c"+CHR$(<color #>)
"c"2""

GO - tell sprite to start
"g"+CHR$(<sprite #>)
"g"2""

HIDE A SPRITE - hide the sprite
"h"+CHR$(<sprite #>)
"h"2""

PAUSE - pause for number of jiffies
"p"+CHR$(<duration>)
"p"2""

SHOW A SPRITE - show the sprite
"s"+CHR$(<sprite #>)
"s"2""

EXPAND WIDTH - expand the width
"x"+CHR$(<expand width?>)
"x"0""

EXPAND HEIGHT - expand the height
"y"+CHR$(<expand height?>)
"y"1""

HALT - stop sprite
animate("")

MOVESPRITE-goto x/y at speed
movesprite(<sprite#>,<x>,<y>,<spd>,<evnt>)
movesprite(2,100,50,120,0)

event bits - meaning

%-------X 1=move now
0=wait for startsprites command
%------X- 1=stop if spritecollision
0=continue regardless
%-----X-- 1=stop if datacollision
0=continue regardless

MOVING -- is sprite moving true/false
moving(<sprite #>)
IF moving(2) THEN

STARTSPRITES --get sprites moving
startsprites
startsprites

STOPSPRITE -- stops specified sprite
stopsprite(<sprite #>)
stopsprite(2)

TURTLE GRAPHICS additions

initialize with: **USE TURTLE**

All the commands in the GRAPHICS package are available from the TURTLE package plus these abbreviations may be used:

BG = BACKGROUND
BK = BACK
CS = CLEARSCREEN
FD = FORWARD
HT = HIDETURTLE
LT = LEFT
PC = PENCOLOR
PD = PENDOWN
PU = PENUP
RT = RIGHT
SETH = SETHEADING
ST = SHOWTURTLE
TEXTBG = TEXTBACKGROUND

More ►

GRAPHICS

initialize with: **USE GRAPHICS**

ARC -- draw an arc
arc(<ctrx>,<ctry>,<rad>,<start angl>,<#deg>)
arc(50,50,20,90,180)

ARCL -- draw arc left from current pos
arcl(<radius>,<angle size>)
arcl(20,180)

ARCR -- draw arc right from current pos
arcr(<radius>,<angle size>)
arcr(30,90)

BACK -- move turtle backwards
back(<length>)
back(45)

BACKGROUND- graphics screen color
background(<color #>)
background(0)

BORDER -- set graphics border color
border(<color #>)
border(15)

CIRCLE -- draw a circle
circle(<centerx>,<centery>,<radius>)
circle(50,50,20)

CLEAR -- clear only graphics viewport
clear
clear

CLEARSCREEN -- clear graphic screen
clearscreen
clearscreen

DRAW -- draw line to x/y offset
draw(<x offset>,<y offset>)
draw(20,30)

DRAWTO-if pendown draw to x/y
drawto(<x coord>,<y coord>)
drawto(50,40)

FILL -- fill in area with current color
(uses non-background pixels as boundary)
fill(<x coord>,<y coord>)
fill(50,40)

FORWARD -- move turtle forward
forward(<length>)
forward(100)

FULLSCREEN -- fullscreen graphics (f5)
fullscreen
fullscreen

GETCOLOR -- returns pixel color
getcolor(<x coord>,<y coord>)
orig'color:=getcolor(50,40)

GRAPHICSCREEN- 0=hi-res/1=multi
graphicscreen(<type>)
graphicscreen(0)

HEADING -- returns turtle heading
heading
turtle'heading:=heading

HOME -- put turtle in its home position
home
home

INQ -- returns graphics screen data
inq(<type function number>)
border(inq(4))

number - meaning
---
0 - graphic screen type
1 - text border color
2 - text background color
3 - text cursor color
4 - graphics border color
5 - graphics background color
6 - current pencolor
7 - textstyle height size
8 - textstyle width size
9 - textstyle direction
10- textstyle overplot?

More ►

11- turtle visible?
12- turtle inside viewport?
13- textscreen displayed?
14- splitscreen active?
15- wrap mode on?
16- pendown?
17- turtle x coordinate
18- turtle y coordinate
19- viewport x minimum
20- viewport x maximum
21- viewport y minimum
22- viewport y maximum
23- window x minimum
24- window x maximum
25- window y minimum
26- window y maximum
27- cosine of heading
28- sine of heading
29- size of turtle

LEFT -- turn turtle left
left(<degrees>)
left(90)

LOADSCREEN-load graphic screen
loadscreen(<filename>)
loadscreen("hrg.horizon")

MOVE-move turtle by offset without line
move(<x offset>,<y offset>)
move(20,30)

MOVETO-moveto x/y point without line
moveto(<x coordinate>,<y coordinate>)
moveto(90,50)

NOWRAP -- won't draw outside viewport
nowrap
nowrap

PAINT -- fill with current color
(uses current pencolor as boundary)
paint(<x coord>,<y coord>)
paint(50,40)

PENCOLOR -- set current pen color
pencolor(<color #>)
pencolor(1)

PENDOWN-turtle draws line if pen down
pendown
pendown

PENUP-turtle won't draw line now
penup
penup

PLOT -- plot a point in current color
plot(<x coordinate>,<y coordinate>)
plot(30,20)

PLOTTEXT-print text on graphics screen
plottext(<x coord>,<y coord>,<text$>)
plottext(10,10,"press f1 now")

PRINTSCREEN-dump graphics to printer
printscreen(<filename>,<offset>)
printscreen("lp:/a+",30)

RIGHT -- turn turtle right
right(<degrees>)
right(90)

SAVESCREEN--save graphics to disk
savescreen(<filename>)
savescreen("hrg.house")

SETHEADING -- set turtle heading
setheading(<degree>)
setheading(0)

SETXY -- set turtle x and y coordinates
setxy(<x coordinate>,<y coordinate>)
setxy(160,0)

SHOWTURTLE -- make turtle visible
showturtle
showturtle

More ►

SPLITSCREEN-4 text lines with graphics
splitscreen
splitscreen

TEXTBACKGROUND-set its color
textbackground(<color #>)
textbackground(0)

TEXTBORDER -- set text border color
textborder(<color #>)
textborder(inq(2))

TEXTCOLOR -- set text cursor color
textcolor(<color #>)
textcolor(1)

TEXTSCREEN -- display text screen (f1)
textscreen
textscreen

TEXTSTYLE -- textstyle for plottext
textstyle(<width>,<height>,<dir>,<overplot>)
textstyle(2,4,0,0)

dir - meaning

-1 - no change
0 - normal (print to the right)
1 - upwards (rotated 90 degrees left)
2 - upside down (print to the left)
3 - downwards (rotated 90 degrees right)

TURTLESIZE -- set turtle size (0 to 10)
turtlesize(<size>)
turtlesize(6)

VIEWPORT -- set drawing frame bounds
viewport(<xmin>,<xmax>,<ymin>,<ymax>)
viewport(0,319,0,199)

WINDOW -- sets the scale of the screen
window(<xmin>,<xmax>,<ymin>,<ymax>)
window(0,319,0,199)

WRAP -- turtle wraps around screen
wrap
wrap

XCOR -- returns x coordinate of turtle
xcor
x:=xcor

YCOR -- returns y coordinate of turtle
ycor
y:=ycor

FONT
initialize with: **USE FONT**

GETCHARACTER-returns char def string
getcharacter(<font#>,<char#>,<string var>)
getcharacter(2,1,char$)

KEEPFONT - make user fonts the default
keepfont
keepfont

LINKFONT --link font to program in mem
linkfont
linkfont

LOADFONT -- load a user-designed font
loadfont(<filename>) //after linkfont
loadfont("font.standard")

PUTCHARACTER -- define a character
putcharacter(<font#>,<char#>,<string var>)
putcharacter(0,1,char$)

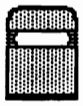
SAVEFONT -- save a user defined font
savefont(<filename>)
savefont("font.charlie")

JOYSTICKS
initialize with: **USE JOYSTICKS**

JOYSTICK -- returns joystick parameters
joystick(<port>,<direction>,<button>)
joystick(1,which'way,fire)

More ►

Sound Effects



PADDLES
initialize with: **USE PADDLES**

PADDLE -- returns paddle parameters
paddle(<port>,<pad1>,<pad2>,<btn1>,<btn2>)
paddle(1,way1,way2,fire1,fire2)

LIGHTPEN
initialize with: **USE LIGHTPEN**

ACCURACY -- set accuracy of lightpen
accuracy(<x range>,<y range>)
accuracy(2,1)

DELAY -- set accuracy time parameter
delay(<time>)
delay(5)

OFFSET -- adjust screen centering
offset(<x correction>,<y correction>)
offset(5,0)

PENON--returns TRUE if pen senses light
penon
IF penon THEN

READPEN -- returns position of pen
readpen(<xcoord>,<ycoord>,<penon?>)
readpen(xpos,ypos,pen'status)

TIMEON -- set pen detection sustain
timeon(<time>)
timeon(3)

DANSK
initialize with: **USE DANSK**

System messages in Danish.

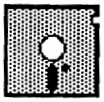
ENGLISH
initialize with: **USE ENGLISH**

System messages in English.
(This is the default language). ■

When we ran this program from Holland, we were astounded by the varity of sounds it can play, and still be so small. Any COMAL 2.0 program can now have different sounds to signal different needs or warnings.

```
// delete "sound'effects"
// save   "sound'effects"
// from Dutch COMAL Users Group
//
PAGE
PRINT "Sound effects demonstration"
PRINT "using ""PROC effect"" with"
PRINT "random numbers 5-254 and 1-30."
PRINT
PRINT "Press STOP to end"
PRINT
RANDOMIZE
USE sound
ZONE 10
PRINT "Effect#","Delay"
PRINT "-------","-----"
TRAP ESC-
REPEAT
  x:=RND(5,254); delay:=RND(1,30)
  PRINT USING " ###": x,
  PRINT USING " ##": delay
  effect(x,delay)
UNTIL ESC
volume(0)
END "Done."
//
PROC effect(x,delay#) CLOSED
  USE sound
  x:=(x+1) MOD 256
  volume(15)
  adsr(1,0,0,15,0)
  soundtype(1,1) //try 1,2 & 1,3
  gate(1,1)
  FOR y:=255 TO 1 STEP -1 DO
    FOR wait#:=1 TO delay# DO NULL
    fr:=x BITAND y
    setfrequency(1,fr*256)
  ENDFOR y
ENDPROC effect ■
```

Kelly's Beach

by Ed Bolton 

Kelly's Beach is an imaginary beach *microworld* which can be populated with people and things by the player. The beach scene is manipulated by first naming an object to be manipulated. There are currently 15 objects:

house	*tree*	*bush*	*cloud*	*sun*
bird	*plane*	*dog*	*boy*	*girl*
pony	*car*	*truck*	*boat*	*fish*

Having named the object, the object may be manipulated by naming one or more action words. There are currently 21 action words:

red	*yellow*	*orange*	*green*	*blue*
purple	*brown*	*small*	*big*	*tall*
wide	*left*	*right*	*up*	*down*
fast	*slow*	*go*	*stop*	*zap*
free				

When the program begins, there is an empty beach scene. In the foreground at the bottom is the ocean, behind (and above) that is the beach followed by a strip of grass, followed by a road, another strip of grass and the sky. Some objects are only allowed in certain parts of the scene; i.e. the boat and the fish can only be in the water; and the car and truck can only be on the road. All other objects can be anywhere but the sky and the water. Objects are named and manipulated by typing the appropriate words. The syntax is very simple. Action words always apply to the last object named. Up to 40 characters may be entered at one time. The only edit keys allowed are cursor left and right, insert and delete. Press the *return key* to pass that line of text to the program for evaluation and action. Example input:

boy *<return>*

A picture of a boy will appear in the scene. His color will be alternating between blue and gray. This alternate flashing between the color selected (blue is the default) and gray indicates the last or current object named. The current object is the one to which all action words apply.

red up *<return>*

The boy will turn red and move up a little (unless moving up would put him into the sky - little boys can't fly).

sun yellow go cloud purple go fast boat big left fast *<return>*

A yellow sun will appear in the sky moving slowly left. A purple cloud will appear in the sky moving rapidly left. When the sun and the cloud reach the left of the display they will go around the *microworld* and reappear at the right edge of the display. Everything else that can 'go' fast or slow will turn and go back the other way when it reaches the edge of the display. The tree, bush and house cannot 'go'; they cannot be set in motion. Some of the laws of physics apply even in a *microworld*. Finally a large blue boat will appear in the water moving rapidly to the left. Actually, this line is more than 40 characters long so it could not be all entered at the same time. The point here is that many words can be entered on the same line - up to 40 characters.

Only eight objects (sprites) are allowed in the beach scene a one time. In this case the laws of COMAL rather than the laws of physics apply. If a ninth object is named, the oldest (first named)

More ►

object is removed before the new one is placed in the scene. This choice won't always be to the players liking. So the action word ZAP is provided. Zap will remove the current object; making room for some other object. Sometimes a player will want to just watch the beach scene he/she has created without having the current object changing color. The action word FREE does that.

If only action words are entered without naming an object (or following free), they are simply ignored. If the action words go, slow, fast or stop are applied to tree, bush or house, they too are simply ignored. On the other hand, if a word is entered which is not in the list of objects or action words, all words up to that word are acted on, then the remainder of the line is displayed again with the offending word in red.

ABOUT THE PROGRAM

The program is table driven so objects and action words can be changed by rebuilding the files *"beachnames.dat"*, *"beachverbs.dat"*, *"beachsprites.dat"*, *"objecttable.dat"* and *"verbtable.dat"*. The file *"beachsprites.dat"* was constructed using *"sprite'editor21"*. The other files were created using *"build'beach'file"*. This program prompts you for data but has no features to re-edit input nor to edit existing files. So be sure you have everything ready before running it. *[A special custom data file creator program is listed in case you don't get Today Disk #12. Run it to create the files needed.]*

Object and action words must be less than or equal to six lower case characters long, left justified in the six character field. In addition to one or two sprite images, objects have associated with them five numeric parameters. They are moves, image'no, lower'limit, upper'limit and object'priority.

moves:

This variable determines how an object will be set into motion.

0=The object can not be set into motion. Example: tree
1=The object will always be set into motion to the left. When it reaches the left edge of the display, it will go around the *microworld* and reenter at the right edge. Example: sun
2=The object will reflect off the edge of the display and go the other way. This type of object always has two sprite images: a left and a right facing image.

image'no:

This is the index into file *"beachsprites.dat"* to locate the sprite image for this object. If moves=2, there are two sprite images. **image'no** points to the first (left facing) image. The right facing image must immediately follow in file *"beachsprites.dat"*.

lower'limit and **upper'limit:**

These establish the smallest and largest values of the Y coordinate allowed for the bottom edge of the image.

object'priority:

With the exception of two objects this variable always has the value -1. One object may be defined to have the lowest priority of all the objects. Its value

More ►

is 7. Example: sun. One other object may be defined to have the highest priority of all the objects. Its value is 0. Example: boat. The priority of all other objects is determined automatically by the program based on the Y coordinate.

Action words are divided into four categories: colors, size, direction and motion. They also must be named for storage in string array verb'name$() in that order. Two action words are defined in procedure evaluate of Kelly's Beach. They are ZAP and FREE. Their names may be changed but the function they perform must be retained. Associated with each action word is a single numeric value, the value of which depends on the category of the action word.

color:

COMAL color code: 2=red, 5=green, etc

size:

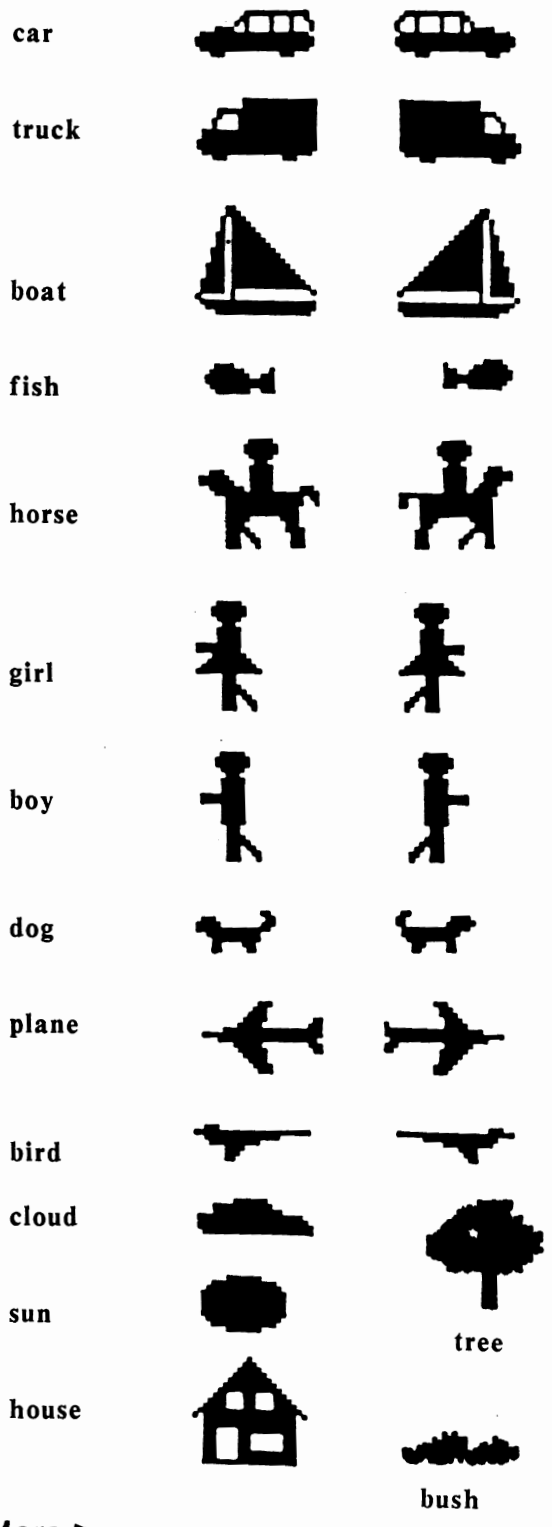
0=neither X nor Y expand
1=y expand only; i.e., tall
2=x expand only; i.e., wide
3=both X and Y expand; i.e., big

direction:

0=left; i.e., lower value of X
1=right; i.e., higher value of X
2=up; i.e., higher value of Y
3=down; i.e., lower value of Y

motion:

0=stop motion. Example: stop
1=set in motion slowly for constant Y.
Example: go slow
2=set in motion rapidly for constant Y.
Example: fast



More ►

Behind the Scenes

Kelly's Beach File Converting

by Captain COMAL

The *Kelly's Beach* program is an excellent system for kids. So good that we wanted to list it in this issue of *COMAL Today*. However, it uses data files - special files on disk that hold information that the program needs. Without these data files, the program will not work.

So we decided to make a program that would create all the data files. To do that, we first had to find out what was in the files ourselves. We used COMAL 2.0 to display all the sprites on the screen. Our program can put 40 sprites on the screen at once. The trick is that we use STAMPSPRITE, so only one sprite is active at any one time. Then we wrote a small 2.0 program that would read the sprite data file and create data statements that can be merged into any COMAL program. We are listing both of these programs here, so you can see how we use COMAL ourselves, here at the COMAL Users Group offices.

Finally, for the special Kelly's Beach data files, we simply read what was in the files, and created our own data statements and a routine to write them to a file. This is included in the Kelly's Beach file creator program also listed in this issue.

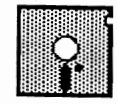
All this work so that all our readers could see what we think is a remarkable program. Is it worth it? Or should long programs and data file dependent programs only be on our *Today Disks*? Listing all the programs in the newsletter takes up lots of pages. This means less room for articles, tips, and notes. Less Questions and Answers. Less letters. Any solution?

DATA STATEMENT MAKER

```
from$:="<insert data file name>"
to'$:="<insert listing file name>"
line:=9000
OPEN FILE 1,from$,READ
OPEN FILE 2,to'$,WRITE
WHILE NOT EOF(1) DO
  PRINT FILE 2: line;"data";
  PRINT FILE 2: ORD(GET$(1,1)),
  FOR x:=2 TO 8 DO
    IF NOT EOF(1) THEN
      PRINT FILE 2: ",",ORD(GET$(1,1)),
    ENDIF
  ENDFOR
  PRINT FILE 2:
  line:+5
ENDWHILE
CLOSE
END "Done."
```

SPRITE SHOW

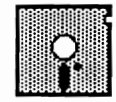
```
// save   "show&stamp"
USE graphics
USE sprites
DIM s$ OF 64
graphicscreen(0)
background(1)
border(1)
pencolor(0)
OPEN FILE 2,"beachsprites.dat",READ
sp:=0; x:=-1
WHILE NOT EOF(2) DO
  x:+1
  READ FILE 2: s$
  define(0,s$)
  identify(0,0); spritecolor(0,0)
  spx:=(x MOD 8)*40
  spy:=(x DIV 8)*40+25
  spritepos(0,spx,spy)
  showsprite(0)
  stampsprite(0)
ENDWHILE
CLOSE■
```

Data File Creator 

Making Kelly's Beach Data Files

```
// delete "0:create'beach"
//  by ed bolton
// save   "0:create'beach"
//
dim dummy$ of 1, s$ of 80
dim ds$ of 80
print chr$(147),chr$(14)
print "This program creates the necessary"
print "files for Kellys Beach"
print
print "Please insert disk you want the files"
input "on and press RETURN: ": dummy$
print
print "Please wait..."
write'beachnames
write'beachverbs
write'objecttable
write'verbtable
write'beachsprites
//
proc read'num
 read num
 write file 2: num
endproc read'num
//
proc read'string
 read s$
 write file 2: s$
endproc read'string
//
proc openfile(name$)
 print "Creating file:";name$
 open file 2,"0:"+name$,write
 ds$:=status$
 if ds$(1:2)<>"00" then
  close
  print "Disk error:";ds$
  stop
 endif
endproc openfile
//
proc closefile
 close file 2
endproc closefile
//
proc write'beachnames
 openfile("beachnames.dat")
 for x:=1 to 2 do read'num
 for x:=1 to 15 do read'string
 closefile
 data 15,25
 data "house","tree","bush","cloud"
 data "sun","bird","plane","dog"
 data "boy","girl","pony","car"
 data "truck","boat","fish"
endproc write'beachnames
//
proc write'beachverbs
 openfile("beachverbs.dat")
 for x:=1 to 4 do read'num
 for x:=1 to 19 do read'string
 closefile
 data 7,4,4,4
 data "red","yellow","blue","green"
 data "orange","purple","brown","small"
 data "big","tall","wide","left"
 data "right","up","down","fast"
 data "slow","go","stop"
endproc write'beachverbs
//
proc write'objecttable
 openfile("objecttable.dat")
 for x:=1 to 75 do read'num
 closefile
 data 0,1,42,135,-1
 data 0,2,42,135,-1
 data 0,3,42,135,-1
 data 1,4,144,170,-1
 data 1,5,144,170,7
 data 2,6,138,170,-1
 data 2,8,138,170,-1
 data 2,10,42,135,-1
 data 2,12,42,135,-1
 data 2,14,42,135,-1
 data 2,16,42,135,-1
 data 2,18,80,103,-1
 data 2,20,80,103,-1
 data 2,22,-5,28,0
 data 2,24,-5,20,-1
endproc write'objecttable
//
proc write'verbtable
 openfile("verbtable.dat")
 for x:=1 to 19 do read'num
 closefile
 data 2,7,6,5,8,4,9,0,3,1,2,0
 data 1,3,2,2,1,1,0
endproc write'verbtable
//
proc write'beachsprites
 openfile("beachsprites.dat")
 while not eod do
  read num
  print file 2: chr$(num),
 endwhile
 closefile
 data 0,64,0,16,0,0,56,0
 data 0,124,0,0,254,0,1,255
 data 0,3,255,128,7,255,192,14
 data 56,224,30,56,240,62,56,248
 data 127,255,252,191,255,250,63,255
 data 248,63,255,248,56,127,248,56
 data 112,56,56,112,56,56,112,56
 data 56,127,248,56,127,248,63,255
 data 248,0,0,64,0,42,128,0
 data 170,128,2,170,0,2,170,160
 data 10,170,160,10,170,168,42,42
 data 170,170,170,170,170,170,168,170
 data 170,170,170,170,136,42,170,168
 data 10,170,160,10,170,160,0,20
 data 0,0,20,0,0,20,0,0
 data 20,0,0,20,0,0,20,0
 data 0,20,0,1,0,64,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,254,0,15,255,0
 data 15,255,192,127,255,252,127,255
 data 254,127,255,255,63,255,255,0
 data 0,64,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,127,0,3,255,224,7,255
 data 240,15,255,248,15,255,248,15
 data 255,248,15,255,248,15,255,248
 data 7,255,240,3,255,224,0,127
 data 0,0,0,64,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,28,0
 data 0,127,255,255,31,255,192,15
 data 252,0,1,224,0,1,192,0
 data 1,128,0,0,0,64,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,28,127,255,255,3,255
 data 248,0,63,240,0,7,128,0
 data 3,128,0,1,128,0,0,64
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,28,0
 data 0,56,0,0,120,0,0,248
 data 3,1,240,7,15,255,255,255
 data 255,255,31,255,254,3,240,7
 data 1,248,3,0,248,0,0,120
 data 0,0,60,0,0,28,0,0
 data 0,64,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 56,0,0,28,0,0,30,0
 data 0,15,0,128,15,128,255,255
 data 240,255,255,255,127,255,248,224
 data 15,192,192,31,128,0,31,0
 data 0,30,0,0,60,0,0,56
 data 0,0,0,64,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,0,0,0
 data 0,0,0,0,0,96,7,0
 data 48,15,128,112,15,255,224,3
 data 255,128,1,255,128,1,135,0
 data 1,131,0,0,0,64,0,0
```

More ►



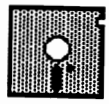

```
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,6,0,0
data 12,1,224,12,3,240,7,255
data 224,7,255,128,3,255,0,1
data 195,0,1,131,0,0,0,64
data 0,62,0,0,127,0,0,127
data 0,0,62,0,0,28,0,0
data 62,0,0,62,0,0,62,0
data 3,254,0,3,254,0,0,62
data 0,0,62,0,0,62,0,0
data 62,0,0,28,0,0,28,0
data 0,30,0,0,31,0,0,29
data 128,0,28,192,0,28,64,0
data 0,64,0,62,0,0,127,0
data 0,127,0,0,62,0,0,28
data 0,0,62,0,0,62,0,0
data 62,0,0,63,224,0,63,224
data 0,62,0,0,62,0,0,62
data 0,0,62,0,0,28,0,0
data 28,0,0,60,0,0,124,0
data 0,220,0,1,156,0,1,28
data 0,0,0,64,0,62,0,0
data 127,0,0,127,0,0,62,0
data 0,28,0,0,62,0,0,62
data 0,0,62,0,3,254,0,3
data 254,0,0,127,0,0,255,128
data 1,255,192,3,255,224,0,28
data 0,0,28,0,0,30,0,0
data 31,0,0,29,128,0,28,192
data 0,28,64,0,0,64,0,62
data 0,0,127,0,0,127,0,0
data 62,0,0,28,0,0,62,0
data 0,62,0,0,62,0,0,63
data 224,0,63,224,0,127,0,0
data 255,128,1,255,192,3,255,224
data 0,28,0,0,28,0,0,60
data 0,0,124,0,0,220,0,1
data 156,0,1,28,0,0,0,64
data 0,62,0,0,127,0,0,127
data 0,0,62,0,0,28,0,0
data 62,0,56,62,0,252,62,0
data 254,62,0,63,190,14,31,255
data 255,7,255,251,7,255,249,7
data 255,240,7,255,240,7,192,240
data 7,128,112,7,192,120,7,96
data 56,7,48,56,7,16,56,0
data 0,64,0,62,0,0,127,0
data 0,127,0,0,62,0,0,28
data 0,0,62,28,0,62,63,0
data 62,63,0,62,124,0,62,248
data 127,255,248,103,255,240,103,255
data 240,7,255,240,7,255,240,7
data 129,240,7,0,240,14,1,240
data 14,3,48,14,6,48,14,4
data 48,0,0,64,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,63,254,0,97,18,0,161
data 17,15,33,17,127,255,254,255
data 255,255,255,255,255,127,255,254
data 28,0,112,0,0,64,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,63,252,0,72,70,0
data 136,69,0,136,69,240,127,255
data 254,255,255,255,255,255,255,127
data 255,254,14,0,56,0,0,64
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,127,255,0,127
data 255,15,255,255,8,255,255,16
data 255,255,16,255,255,127,255,255
data 127,255,255,255,255,255,255,255
data 255,127,255,254,28,0,112,0
data 0,64,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,255,255,0
data 255,255,0,255,255,224,255,255
data 144,255,255,136,255,255,136,255
data 255,254,255,255,254,255,255,255
data 255,255,255,127,255,254,14,0
data 56,0,0,64,3,0,0,3
data 128,0,5,192,0,5,224,0
data 5,240,0,13,248,0,13,252
data 0,13,254,0,29,255,0,29
data 255,128,29,255,192,61,255,224
data 61,255,240,61,255,248,125,255
data 252,125,255,254,125,0,1,129
data 0,0,127,255,255,63,255,255
data 15,255,254,0,0,64,0,0
data 192,0,1,192,0,3,160,0
data 7,160,0,15,160,0,31,176
data 0,63,176,0,127,176,0,255
data 184,1,255,184,3,255,184,7
data 255,188,15,255,188,31,255,188
data 63,255,190,127,255,190,128,0
data 190,0,0,129,255,255,254,255
data 255,252,127,255,240,0,0,64
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,60,0
data 0,127,4,0,255,156,0,255
data 252,0,127,252,0,63,156,0
data 0,64,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 30,0,16,127,0,28,255,128
data 31,255,128,31,255,0,28,254
data 0,0
endproc write'beachsprites
```

Joysticks Paddles

Previously we published routines to read the joystick and paddle ports. Later the cartridge came out with its own routines. The following two COMAL 0.14 procedures emulate the **joystick** and **paddle** commands of the COMAL 2.0 cartridge.

```
proc paddle(port,ref x,ref y,ref fire1,
ref fire2) closed // wrap line
  cia:=56320; sid:=54272
  poke cia+13,1 //disable timer a intrpt
  ddra:=peek(cia+2)
  poke cia+2,192
  poke cia,64*port
  x:=peek(sid+25); y:=peek(sid+26)
  poke cia+2,ddra
  poke cia+13,129 //enable timer
  mem:=peek(cia+2-port)
  fire1:=1-((mem mod 16) div 8)
  fire2:=1-((mem mod 8) div 4)
endproc paddle

proc joystick(port,ref direction,ref fire)
closed // wrap line
  if port=1 then //          1
    mem:=peek(56321) //  8    2
  elif port=2 then //     7   0   3
    mem:=peek(56320) //  6    4
  else //                   5
    return // not proper port number
  endif
  fire:=1-((mem mod 32) div 16)
  // fire = true if pressed
  case 15-(mem mod 16) of
  when 1
    direction:=1
  when 2
    direction:=5
  when 4
    direction:=7
  when 5
    direction:=8
  when 6
    direction:=6
  when 8
    direction:=3
  when 9
    direction:=2
  when 10
    direction:=4
  otherwise
    direction:=0
  endcase
endproc joystick
```

Kelly's Beach



The Program Listing

```
// kelly's beach by ed bolton
dim c$ of 1, s$ of 64, cw$ of 6
dim sprite'table(0:7,8)
//
proc define'sprite
 tmp9:=sprite'table(i,sprite'no)
 tmp8:=object'table(sprite'table(i,object'no),image'no)
 identify tmp9,tmp8+sprite'table(i,s'direction)
 spritecolor tmp9,sprite'table(i,s'color)
 if sprite'table(i,s'size)=1 or sprite'table(i,s'size)=3 then
  tmp8:=42
 else
  tmp8:=21
 endif
 spritepos tmp9,sprite'table(i,x'coord),sprite'table(i,y'coord)+t
 mp8 // wrap line
 tmp8:=sprite'table(i,s'size)=1 or sprite'table(i,s'size)=3
 spritesize tmp9,sprite'table(i,s'size)>1,tmp8
endproc define'sprite
//
proc select'sprite'numbers
 x0:=0; x1:=sn-1; tmp1:=-1
 for i:=0 to sn-1 do
  x:=object'table(sprite'table(i,object'no),object'priority)
  y:=sprite'table(i,sprite'no)
  if x>=0 then
   if x=0 then tmp1:=0
   if y<>x then
    if y>=0 then hidesprite y
    sprite'table(i,sprite'no):=x
   endif
   define'sprite
   if i=x0 then x0:+1
   if i=x1 then x1:-1
  else
   if y>=0 then
    hidesprite y
    sprite'table(i,sprite'no):=-1
   endif
  endif
 endfor i
 if x0<=x1 then
  tmp1:+1
  repeat
   tmp2:=false; y:=1000
   for j:=x0 to x1 do
    if sprite'table(j,sprite'no)=-1 and sprite'table(j,y'coord)<y t
    hen // wrap line
     tmp2:=true; i:=j
     y:=sprite'table(j,y'coord)
    endif
   endfor j
   if tmp2 then
    sprite'table(i,sprite'no):=tmp1
    define'sprite
    tmp1:+1
    if i=x0 then x0:+1
    if i=x1 then x1:-1
    if x1<x0 then tmp2:=false
   endif
  until tmp2=false
 endif
endproc select'sprite'numbers
//
proc select'object
 tmp2:=false; j:=0
 while j<=sn-1 and tmp2=false do
  if sprite'table(j,object'no)=i then tmp2:=true
  j:+1
 endwhile
 if tmp2=true then
  cs:=j-1
 else
  tmp1:=i
  if sn=8 then
   cs:=0
   zap'current'sprite
  endif
  sn:+1; cs:=sn-1
  sprite'table(cs,object'no):=tmp1
  sprite'table(cs,sprite'no):=-1
  sprite'table(cs,speed):=0
  sprite'table(cs,x'coord):=rnd(-10,300)
  tmp2:=rnd(object'table(tmp1,lower'limit),object'table(tmp1,up
  per'limit)) // wrap line
  sprite'table(cs,y'coord):=tmp2
  sprite'table(cs,s'direction):=0
  sprite'table(cs,s'color):=6
  sprite'table(cs,s'size):=0
  select'sprite'numbers
 endif
endproc select'object
//
proc objects
 i:=1
 repeat
  if object'name$(i)=cw$ then
   not'found:=false
  else
   i:+1
  endif
 until not'found=false or i>no'of'objects
 if not'found=false then
  if cs<>-1 then
   if sprite'table(cs,object'no)<>i then
    spritecolor sprite'table(cs,sprite'no),sprite'table(cs,s'color)
    select'object
   endif
  else
   select'object
  endif
 endif
endproc objects
//
proc move'left'right
 sprite'table(j,x'coord):=sprite'table(j,x'coord)+distance
 if distance<0 then
  if sprite'table(j,x'coord)<-10 then
   if object'table(sprite'table(j,object'no),moves)=2 then
    sprite'table(j,s'direction):=1
    tmp8:=true
   endif
   if object'table(sprite'table(j,object'no),moves)=1 then
    sprite'table(j,x'coord):=300
   else
    sprite'table(j,x'coord):=-10
```

More ►

```
  endif
 endif
else
 if sprite'table(j,x'coord)>300 then
  if object'table(sprite'table(j,object'no),moves)=2 then
   sprite'table(j,s'direction):=0
   tmp8:=true
  endif
  if object'table(sprite'table(j,object'no),moves)=1 then
   sprite'table(j,x'coord):=-10
  else
   sprite'table(j,x'coord):=300
  endif
 endif
endif
tmp9:=sprite'table(j,sprite'no)
if tmp8=true then
 tmp8:=object'table(sprite'table(j,object'no),image'no)
 identify tmp9,tmp8+sprite'table(j,s'direction)
endif
if sprite'table(j,s'size)=1 or sprite'table(j,s'size)=3 then
 tmp8:=42
else
 tmp8:=21
endif
spritepos tmp9,sprite'table(j,x'coord),sprite'table(j,y'coord)+t
mp8 // wrap line
endproc move'left'right
//
proc verbs
 i:=1
 repeat
  if verb'name$(i)=cw$ then
   not'found:=false
   if cs<>-1 then
    if i<i'size then
     sprite'table(cs,s'color):=verb'table(i)
     tmp9:=sprite'table(cs,sprite'no)
     spritecolor tmp9,sprite'table(cs,s'color)
    else
     if i<i'dire then
      sprite'table(cs,s'size):=verb'table(i)
      tmp9:=sprite'table(cs,sprite'no)
      tmp8:=sprite'table(cs,s'size)=1 or sprite'table(cs,s'size)=3
      spritesize tmp9,sprite'table(cs,s'size)>1,tmp8
      if sprite'table(cs,s'size)=1 or sprite'table(cs,s'size)=3 then
       tmp8:=42
      else
       tmp8:=21
      endif
      spritepos tmp9,sprite'table(cs,x'coord),sprite'table(cs,y'coo
      rd)+tmp8 // wrap line
     else
      if i<i'motion then
       if verb'table(i)<2 then
        if object'table(sprite'table(cs,object'no),moves)=2 then
         sprite'table(cs,s'direction):=verb'table(i)
        endif
        distance:=10*(2*verb'table(i)-1)
        j:=cs; tmp8:=true
        move'left'right
       else
        if verb'table(i)=3 then
         sprite'table(cs,y'coord):=sprite'table(cs,y'coord)+10
         tmp9:=object'table(sprite'table(cs,object'no),upper'limit)
         if sprite'table(cs,y'coord)>tmp9 then
          sprite'table(cs,y'coord):=tmp9
         endif
        else
         sprite'table(cs,y'coord):=sprite'table(cs,y'coord)-10
         tmp9:=object'table(sprite'table(cs,object'no),lower'limit)
         if sprite'table(cs,y'coord)<tmp9 then
          sprite'table(cs,y'coord):=tmp9
         endif
        endif
        select'sprite'numbers
       endif
      else
       if object'table(sprite'table(cs,object'no),moves)<>0 then
        sprite'table(cs,speed):=verb'table(i)
       endif
      endif
     endif
    endif
   endif
  endif
  i:+1
 until not'found=false or i>no'of'verbs
endproc verbs
//
proc update'beach
 c'slow:=(c'slow+1) mod n'slow
 c'blink:=(c'blink+1) mod n'blink
 if c'blink=0 then
  if cs<>-1 then
   t'blink:=(t'blink+1) mod 2
   if t'blink=0 then
    spritecolor sprite'table(cs,sprite'no),12
   else
    spritecolor sprite'table(cs,sprite'no),sprite'table(cs,s'color)
   endif
  endif
 endif
 tmp1:=2
 if c'slow=0 then tmp1:=1
 if sn<>0 then
  for j:=0 to sn-1 do
   if sprite'table(j,speed)>=tmp1 then
    distance:=2*sprite'table(j,s'direction)-1
    tmp8:=false
    move'left'right
   endif
  endfor j
 endif
endproc update'beach
//
proc read'sprites
 open file 2,"0:beachsprites.dat",read
 for i:=1 to no'of'sprites do
  read file 2: s$
  define i,s$
 endfor i
 close
endproc read'sprites
//
proc read'words
```

More ►

```
 open file 2,"0:beachnames.dat",read
 read file 2: no'of'objects
 dim object'table(no'of'objects,5)
 dim object'name$(no'of'objects) of 6
 read file 2: no'of'sprites
 for i:=1 to no'of'objects do
  read file 2: s$
  object'name$(i):="      "
  object'name$(i)(1:len(s$)):=s$
 endfor i
 close
 open file 2,"0:beachverbs.dat",read
 read file 2: no'of'colors
 read file 2: no'of'size
 read file 2: no'of'direction
 read file 2: no'of'motion
 no'of'verbs:=no'of'colors+no'of'size+no'of'direction+no'of'moti
 on // wrap line
 dim verb'name$(no'of'verbs) of 6
 dim verb'table(no'of'verbs)
 for i:=1 to no'of'verbs do
  read file 2: s$
  verb'name$(i):="      "
  verb'name$(i)(1:len(s$)):=s$
 endfor i
 close
 i'size:=no'of'colors+1
 i'dire:=i'size+no'of'size
 i'motion:=i'dire+no'of'direction
endproc read'words
//
proc read'object'table
 open file 2,"0:objecttable.dat",read
 for i:=1 to no'of'objects do
  for j:=1 to 5 do
   read file 2: object'table(i,j)
  endfor j
 endfor i
 close
endproc read'object'table
//
proc read'verb'table
 open file 2,"0:verbtable.dat",read
 for i:=1 to no'of'verbs do
  read file 2: verb'table(i)
 endfor i
 close
endproc read'verb'table
//
proc waves
 x:=0; y:=40
 while x<320 do
  if y<40 then
   y:=40; x:+12
  else
   y:=32; x:+4
  endif
  drawto x,y
 endwhile
endproc waves
//
proc beach
 background 1
 setgraphic 0
 hideturtle
 band(135,104,13)
 band(103,80,15)
 band(79,56,13)
 penup
 setxy 0,40
 pencolor 0
 pendown
 waves
 pencolor 14
 fill 100,0
endproc beach
//
proc band(y0,y1,c) closed
 penup
 pencolor c
 setxy 0,y0
 pendown
 drawto 320,y0
 penup
 setxy 0,y1
 pendown
 drawto 320,y1
 fill 100,y1+1
endproc band
//
proc clear'input'string
 s$:=""; cp:=1 // input char pos
 for i:=1 to 40 do s$:=s$+" "
endproc clear'input'string
//
proc zap'current'sprite
 if cs>=0 then
  hidesprite sprite'table(cs,sprite'no)
  if sn>1 then
   if sn-1>cs then
    for i:=cs+1 to sn-1 do // Close up table
     for j:=1 to 8 do
      sprite'table(i-1,j):=sprite'table(i,j)
     endfor j
    endfor i
   endif
  endif
  cs:=-1 // No current sprite
  sn:=sn-1 // 1 less active sprite
 endif
endproc zap'current'sprite
//
proc evaluate
 s$(41:41):=chr$(13)
 s$(42:42):=chr$(13)
 error:=false; cp:=1
 repeat
  while s$(cp:cp)=" " do cp:+1
  icp:=cp
  repeat
   cp:+1
  until s$(cp:cp)=" " or s$(cp:cp)=chr$(13)
  if s$(icp:cp)<>chr$(13)+chr$(13) then
   if cp-icp<=6 then
    cw$:="      "
    cw$(1:cp-icp):=s$(icp:cp-1)
```

More ►

```
    not'found:=true
    objects
    if not'found then verbs
    if not'found then
     if cw$="free  " then
      not'found:=false
      if cs>=0 then
       spritecolor sprite'table(cs,sprite'no),sprite'table(cs,s'color)
       cs:=-1
      endif
     else
      if cw$="zap   " then
       not'found:=false
       zap'current'sprite
      endif
     endif
    endif
    if not'found then error:=true
   else
    error:=true
   endif
  endif
 until error or s$(cp:cp)=chr$(13)
 if error then
  cp:=cp-icp
  if icp>1 then
   s$:=s$(icp:40)
   for i:=len(s$)+1 to 40 do s$:=s$+" "
  endif
  plottext 8*(ai-1),187," "
  pencolor 2
  plottext 0,192,s$(1:cp)
  pencolor 11
  plottext 8*cp,192,s$(cp+1:40)
  cp:=1
  plottext 8*(cp-1),187,chr$(94)
  ai:=cp
 else
  clear'input'string
 endif
endproc evaluate
//
// Main program segment
read'words
read'sprites
read'object'table
read'verb'table
beach
sn:=0 // Number of active sprites
cs:=-1 // Current sprite pointer; neg=none
clear'input'string
print chr$(14) //lower case
//
// Index names for object'table
moves:=1 // 0=no, 1=wrap, 2=reflect
image'no:=2
lower'limit:=3; upper'limit:=4
object'priority:=5 // Negative means none
//
// Index names for sprite'table
object'no:=1; sprite'no:=2
speed:=3 // 0=stop, 1=slow, 2=fast
x'coord:=4; y'coord:=5
s'direction:=6 // 0=left, 1=right
s'color:=7
s'size:=8 // 0=nominal, 1=tall, 2=wide, 3=big
//
spriteback 9,3
pencolor 11
c'slow:=0; c'blink:=0; t'blink:=0
n'slow:=5; n'blink:=10; ai:=1
plottext 0,187,chr$(94)
//
// main program loop
while true do // i.e., do forever
 c$:=key$
 if c$<>chr$(0) then
  error:=false
  if "a"<=c$ and c$<="z" or c$=" " then // replace
   s$(cp:cp):=c$; cp:+1
   if cp>40 then cp:=40
  else
   case c$ of
   when chr$(20) // delete
    if cp>1 then
     if cp=2 then
      s$:=s$(2:40)+" "
     else
      s$:=s$(1:cp-2)+s$(cp:40)+" "
     endif
     cp:-1
    endif
   when chr$(148) // insert
    if cp=1 then
     s$:=" "+s$(1:40)
    else
     s$:=s$(1:cp-1)+" "+s$(cp:40)
    endif
   when chr$(29) // cursor right
    if cp<40 then cp:+1
   when chr$(157) // cursor left
    if cp>1 then cp:-1
   when chr$(13) // return
    evaluate
   otherwise
   endcase
  endif
  if error=false then
   plottext 8*(ai-1),187," "
   plottext 0,192,s$
   plottext 8*(cp-1),187,chr$(94)
   ai:=cp
  endif
 endif
 update'beach
endwhile
```

Dear COMAL Readers- We at *Ahoy!* have been aware of COMAL for some time now but it wasn't until we were browsing through the various SIGS on the PlayNet system that we realized how popular the language had become. Our interest in COMAL has now grown to the point where we are considering publishing various COMAL articles and programs in our magazine. Len Lindsay and I both feel that this would be a major leap in the advancement of COMAL, but we need your help to make it a reality. Before *Ahoy!* can publish COMAL articles and programs we want to be assured that we will be meeting the needs of our readers. If you would like to see COMAL articles and programs published in *Ahoy!* magazine, please write to us at this address:

Ahoy!/COMAL
45 W 34 St, Suite 407
New York, NY 10001

If the amount of cards and letters we receive indicates that we would best be serving the interest of our readers by publishing COMAL programs and articles then we will start doing so as soon as possible. B. W. Behling, Ahoy!

This is your chance to really help COMAL. Most of you can't afford to subscribe to every major computer magazine, so we might as well pick one good one (from the "big four"). Ahoy! is already publishing several COMAL programs on every monthly Ahoy! disk, beginning with June 1986. They are supporting COMAL. Let's support them. Please send a letter or post card to them explaining why they should publish COMAL articles. Keep a copy of it - it will be valuable later! And yes, we also support "the little guys" like Info (ratings), Transactor (technical) and Guide (general).

Dear Ahoy Editor: Although I do not subscribe to *Ahoy*, I own every issue since the first. I find much to like about it and look forward to my trip to Harvard Square to pick up the latest issue. I do subscribe to *Compute*, its *Gazette*, *RUN* and *InfoWorld*. I always buy *Byte* and *Creative Computing*. I use my C64 for wordprocessing, developing college level CAI, keeping a mailing list for a club and telecommunicating. I may not be typical of your readership, but may be typical of some segment thereof. It is thus not only for myself that I write.

In the September Scuttlebut you promised an eight page section devoted to a previously neglected area of home computing. When I read that, I thought *"It will never be, but wouldn't it be great if there were a section devoted to COMAL and the truly professional power that the now inexpensive cartridge version provides the 'lowly' C64 owner."* If by some miracle that IS what you are planning, then accept my heartfelt congratulations on your far-reaching vision, and skip the rest of this letter. If not, please read on and give me the chance to convince you that a monthly 8-10 pages devoted to COMAL would be in your best interest.

I will soon be letting my magazine subscriptions lapse. They just seem to repeat themselves and provide the same routines and utilities. Since I use COMAL for any programming I want to do, they just don't earn their keep in my budget. So one reason for you to support COMAL is readers like me, who wouldn't dream of using Microsoft BASIC 2.0 when for the price of a few games we can get into a system that has tremendous advantages and only illusory disadvantages.

I will not tell you about the COMAL cartridge, since I assume you are well

aware of it. I'm sure that if every C64 in the world came with one installed, any competent computer person would **not hesitate** for an instant. That COMAL is inherently superior to BASIC 2.0 is not really at question, is it? BASIC's advantages come down to one: it is ubiquitous. But since Commodore BASIC does not port to an IBM PC while COMAL 2.0 does, even that advantage is illusory. Besides, the nearly free COMAL 0.14 can be included on any C64, and with the free fastloader you get into it in a hurry and have plenty of RAM for some serious applications. (Would one program in *Ahoy* not fit?) Since COMAL is actually more portable than BASIC rather than less so, it offers to help *Ahoy*. I'm keeping my C64 for awhile, but the Amiga sure looks good. How many pages will you devote to each machine? How will you serve the new fragmented reader population? You've probably guessed the point: with Amiga COMAL on the way, you have an easy way to produce some lovely glossy pages that will interest owners of all three machines. What good capitalist can fail to see the value in that?

There is an added bonus. As you begin to publish COMAL code, any re-hashing you do for those readers new to your pages, will become new and interesting for your older readers, too. I'd **love** to read Orson Scott Card's articles about games using SETSPRITE and ANIMATE rather than those gowdawful pokes. I highly value his clear accounts of how it is done in BASIC because the POKEs show me the registers and bits and allow me to get a sense of what is going on at the hardware level, but actually I most want to drive the car, not tinker with its innards. COMAL lets me do all that stuff (and more) just as fast, and with less time needed for learning extraneous details. Orson could then do more of what he does so well; talk about games themselves at the human level.

Wait. That's not all. You would be helping to produce young programmers with a better chance to survive Pascal in school and a better basis for serious work in computer science later on. Some would even say it is your patriotic duty to support top-down modular programming, since America needs programmers who can work in teams using structured, modular code.

I am certain that as you begin to support COMAL, your readers will see what it looks like and how it works and learn about the programming environment that it provides. More and more of them will use COMAL, especially those thinking of how they can write on their C64s code that they can use on almost any machine they upgrade to.

I don't know if this convinces you, but After teaching myself BASIC, Logo, and PILOT, I still found serious barriers in doing the kind of programming I wanted to do. With COMAL, I can actually put together my own development package for writing CAI, simply by using or adapting existing procedures. I can't tell you what a thrill it is for an amateur like me to realize that he can create his own COMAL version of PILOT, that will allow users PILOT's convenience with none of its restrictions (since they will be working with a superset of COMAL and, hence, have access to the full power of the machine).

COMAL is catching on fast, because it deserves to. By publishing a COMAL section you will insure that a growing number of us continue to find *Ahoy* worth a trip to the newsstand. - Jim Ventola, Roslindale, MA

[This is the letter Mr. Ventola sent to Ahoy. Borrow from it, or pull paragraphs out of COMAL Today for <u>your</u> letter. You have our permission! Save a copy, it will be worth something soon (hint).]

Questions and Answers

PRINTER OUTPUT

Question: How do I have my program print to my printer? - Bob McCauley, NY

Answer: It is easy to switch a program's output to go to the printer instead of the screen. Add this line:

```
select output "lp:"
selçct "lp:"        //<---short form
```

All PRINT statements will now send their output to the printer instead of the screen. To get the output back to the screen issue this command:

```
select "ds:"
```

Note: PRINT statements are affected by the SELECT command. INPUT statements are not affected. An INPUT prompt will appear on the screen even if the printer has been selected as the output device. ■

ALTER A STRING

There are times when you would like to make sure that certain characters were not in a string variable BEFORE you use it. One case where this is important is when using a variable with a PRINT USING format specification. The variable should not contain unwanted "#" characters since it is has a special meaning. The following can be used to change an unwanted character into another character (change "#" into "%"):

```
proc alter(ref text$) closed
  repeat
    if "#" in text$ then
      text$("#" in text$):="%"
    endif
  until not ("#" in text$)
endproc alter
```
■

EMPTY SOCKET

Question: Reference to an empty socket on the COMAL 2.0 Cartridge into which EPROM's may be plugged are made in the literature yet no such socket exists on my cartridge. Is my cartridge an older version or what? - William Staneski, Suffolk, VA

Answer: The first 2,000 COMAL 2.0 cartridges used 4 EPROM's to hold the 64K of code for the COMAL 2.0 system. After that, two ROM's hold the same 64K of code. A note was included with the first cartridges advising owners that an empty socket was not included due to lack of space on the cartridge board. If you cartridge is black, you have an empty socket inside. To access it you must open the cartridge (voiding the waranty). A properly prepared EPROM can be installed in the socket. This is not for novices! ■

REAL OR INTEGER?

Surprisingly, integer math takes longer to perform than floating point math in COMAL. This is because COMAL translates integers to floating point numbers, and does all math as floating point. Therefore, to gain a small increase in speed of execution, use real variables and constants. While these take up more room, they will execute faster.

To use real constants, just include a decimal point. Thus, 3 becomes 3.0, etc. However, when the line is listed, the number will be displayed as 3 (an integer), so be carefull when re-editing these lines. Also note that this limit does not apply to FOR loops. A FOR loop with an integer counter executes faster than one with a real counter. ■

Questions and Answers

COLOR SWITCHING

A COMALite on PlayNet asked if an input prompt and the reply could be in two different colors. What a wonderful idea to allow you to quickly see what you've typed just by color. And it can be done in many ways. One example follows (use with a gray background):

```
print chr$(147), // clearscreen
dim name$ of 10, ok$ of 1
prompt("Name:",name$)
prompt("OK?",ok$)
end
//
proc prompt(text$,ref reply$) closed
  print chr$(144),text$, //144=black
  input chr$(5):reply$ //5=white
endproc prompt
```

ROD THE ROADMAN SOLUTION

Travis Lappe has presented us with the following solution to the final *Rod the Roadman* problem in *COMAL Today #9*, how to get out of any house:

```
PROC get'out
  LOOP
    IF clear'ahead THEN
      move
    ELSE
      lt // left
      move
      right'check
    ENDIF
    get'out
  ENDLOOP
ENDPROC get'out
//
PROC right'check
  IF clear'right THEN
    rt // right
    move
  ENDIF
ENDPROC right'check
```

REMOTE CONTROLLED C64

Dear COMAL User's Group- I would like to connect an external terminal to my C64 running COMAL. In other words, I want to call my C64 via a modem (from our mainframe at work), have the C64 answer with its modem, and then do all my COMAL programming and commands from the remote terminal. I can use SELECT:

```
select output "sp:"
select input "sp:"
```

But that doesn't quite do it. Error messages are directed to the C64 itself regardless of the output selection. Is there a way to direct ALL output to the serial port? Also, what do I need to send the C64 to make it think the *STOP* key has been pressed? - David Funk, 3372 Victoria Ave, Lafayette, CA 94549

We don't know how to do this. We included your address in case some COMALite knows how to do this. Try sending a CONTROL C or a CHR$(3) to simulate the STOP key. Related information:

Modem Fun With COMAL 2.0, COMAL Today #9, page 10
Connect Your C64 To An IBM, COMAL Today #9, page 12

PAPERCLIP FONT

If you use the PaperClip wordprocessor, you will be happy to know that many of the fonts on our *FONT DISK* will work with it. Any font whose file name starts with "set." and ends with ".b" will work. Use this PaperClip command followed by the font file name:

CONTROL ↑

Power Supply Notes

by Jay Renard 

reprinted from Comm'putoy Cult, Aug 85

How many peripherals can you plug into your C64 before your power supply is pushed to the limit? Note that disk drives, monitors, and printers do not draw power directly from the C64. Interfaces, cartridges and modems **do** draw power directly from your computer power supply. How much your supply can draw depends on what it is rated at. Looking at my power supply it says:

Output 5VDC 7.5VA;
9VAC 9.0VA.

Translated that means to divide 7.5VA by 5VDC to get maximum **DC** current the supply can handle. Divide 9.0VA by 9VAC to get the maximum **AC** current the supply can handle. Doing the arithmetic, my supply can handle a maximum of 1.5 amps DC and 1 amp AC:

7.5VA/5VDC = 1.5ampDC max DC current
9.0VA/9.0VAC = 1ampAC max AC current

Running at maximum ratings increases the chance of early power supply failure. How much current does your system draw? The C64 by itself needs:

.8amp DC and .56amp AC.

Now add in the power demands of various plug ins and their current ratings (sorry, but we don't know the power rating of the cartridges. Commodore promised to tell us, but we're still waiting after months):

Modems		
64modem	-	.042
MFG Direct Connect	-	.014
Hes Modem	.020	.064

Interfaces	DC amp	AC amp
Buscard II	.55	-
BI-80	.60	-
Turboprint-GT/32k buf	.30	-
Turboprint-GT	.20	-
Cardco 5 card exp.	.10	-
Cardco +G	.10	-
CardcoPS	.05	-
64 Link	.05	-
Paperclip dongle	.05	-

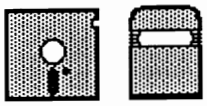
Cartridges		
Fastload	.12	-
Pinball	.12	-
Mach 5	.10	-
Tele easy	.10	-
Calc result	.10	-
Currah speech 64	.10	-

This is a cross section of available plug-ins for the C64. Similar products will draw about the same current. You can estimate what your system is drawing by useing the chart and comparing the nearest product to what you have. For example:

	DC amp	AC amp
Buscard II,	.55	-
Cardco +G	.10	-
a 64 Modem(Westridge	-	.042
C64 computer	.80	.560
Total usage	1.45 amp	.602 amp

The AC draw is well within limits but the DC current load is close to maximum limits. Power supply failure depends on how heavy the system use is. Occasional use will prolong the power supply life. The above described system indicates moderate to heavy usage. Evaluate your system needs, your system peripherals, and your power supply's capability to handle your system. *[Ed note: can someone send us a sequel: including the 2.0 Cartridge?]* ■

C128 / 1571 Page



1571 AS DUAL DRIVE

The 1571 disk drive from Commodore can use both sides of a disk, or act like a 1541 and use just one. Which side the drive will use can be switched under program control. If the 1571 were set to act like a 1541 (using only one side of a disk), you could have a directory on each side, giving double the number of directory entries normally possible. First we have to switch the drive into 1541 mode:

```
PASS "U0>M0"
```

Now the 1571 will act as a 1541 and use only one side of the disk. Since we plan on using each side of the disk separately, we have to format the disk twice (once for each side - don't remove the disk - this is automatic):

```
PASS "U0>H1"              // front
PASS "N0:DISK NAME,ID"

PASS "U0>H0"              // back
PASS "N0:SIDE 2 NAME,ID"
```

Now the disk is formated to look like two different disks. Unfortunately you cannot use COMAL's normal method of referring to two drives ("0:" and "1:"), but you can switch which side of the disk is *"active"* with a PASS command.

To turn the 1571 back into its normal double sided mode, you PASS it the following command:

```
PASS "U0>M1"     // 1571 normal
```

The following procedures can be used in your programs:

```
PROC single'sided // 1541 mode
  PASS "u0>m0"
ENDPROC single'sided
//
PROC double'sideded // 1571 mode
  PASS "u0>m1"
ENDPROC double'sided
//
PROC front'side // while in 1541 mode
  PASS "u0>h1"
ENDPROC front'side
//
PROC back'side // while in 1541 mode
  PASS "u0>h0"
ENDPROC back'side
```

C128 FAST MODE

The 8502 processor in the C128 can be run at two speeds. By changing a new register in the VIC chip of the C128, we can double the clock speed. While in *"FAST"* mode, the 40 column screen will be switched off (the VIC chip doesn't have enough time to access memory). BASIC 7.0 has the commands **FAST** and **SLOW**, so we will too:

```
PROC fast
  POKE 53296,3
ENDPROC fast
//
PROC slow
  POKE 53296,0
ENDPROC slow
```

C128 WARNING

It seems that the original COMAL 2.0 grey cartridges will not work with the C128, but the newer black cartridges do. It must be due to power requirements, since the grey cartridges need over twice as much power. ■

Today Disk #11 - Front

66 Files — 4 Blocks Free:

boot c64 comal	file.5.j	----------	calvin.hrg	- send sase to -
c64 comal 0.14	file.6.j	dump.1520	natalie.hrg	----------
comalerrors	file.7.j	dump.1525	----------	- comal users -
ml.sizzle	file.8.j	dump.bx80	-compacted pix -	- group, usa -
hi	file.9.j	dump.epson	----------	- 6041 monona -
----------	file.10.j	dump.imp	compact pix	- madison, wi -
-support files -	file.11.j	dump.nec	lightbulb.crg	- 53716 -
----------	file.12.j	dump.nec.b	objects.crg	----------
- do not load -	file.13.j	dump.oki92	----------	-(608)222-4432 -
----------	file.14.j	dump.oliv	- graphics -	----------
file.1.j	file.15.j	----------	- editor by -	
file.2.j	file.16.j	- bitmap pix -	-colin thompson-	
file.3.j	----------	----------	----------	
file.4.j	- screen dumps -	directory	-for more info -	

Today Disk #11 - Back

95 Files — 1 Blocks Free:

hi	live'menu/demo	----------	proc.oki'dump	----------
----------	old'english'2.0	set.old'english	proc.real'fft	pkg.dobbelskaerm
- comal 2.0 -	roll'dice.pop	----------	proc.test'signal	pkg.text
- programs -	sort'demo	- compacted -	----------	----------
----------	statistics/demo	- pictures -	- listed -	- source code -
boxes.pop	----------	----------	- programs -	----------
c64mode80col	- comal 0.14 -	compact pix	----------	src.text
correct'disk	- program -	chip.crg	lst.base'conv	----------
demo/dualscreen	----------	keyboard.crg	lst.delink	-for more info -
demo/text1	old'english'.14	liberty.crg	lst.graphs	- send sase to -
demo/text2	----------	----------	----------	----------
differentiat.pop	- data files -	- comal 2.0 -	-popover system-	- comal users -
fft'test.pop	----------	- procedures -	----------	- group, usa -
fix'disk.pop	- do not load -	----------	pop.skeleton	- 6041 monona -
flash'cursor	----------	proc.backup	pop.colors	- madison, wi -
fun'print.pop	ml.greymat	proc.cursor'off	pop.dir	- 53716 -
greymat	dat.cards	proc.cursor'on	pop.dir+colors	----------
inq'example.pop	----------	proc.fft	----------	-(608)222-4432 -
live'menu.pop	- basic font -	proc.fix'modem	- packages -	----------

Today Disk #11 Contributors

Christopher Abissi

Phyrne Bacon

Marcel Bokhorst

Dick Klingens

Tom Kuiper

Len Lindsay

Susan Long

David Powell

Joel Rea

Terry Rickets

David Stidolph

Ken Strahl

Colin Thompson

HOW TO TYPE IN PROGRAMS

Line numbers are required for your benefit in editing a program (but are irrelevant to a running program). Thus most magazines do not use line numbers when listing a COMAL program. It is up to **YOU** to provide the line numbers. Of course, **COMAL can do it for you.** Follow these steps to enter a COMAL program:

1) Enter command: **NEW**
2) Enter command: **AUTO**
3) Type in the program
4) When done:
   Version 0.14: Hit *<return>* key twice
   Version 2.0 : Hit *<STOP>* key

While entering a program, use unshifted letters. If letters are capitalized in the listing it does not mean to use SHIFT with those letters. They are capitalized merely to be easy to read. The only place to use SHIFTED letters is inside quotes. Also, you don't have to type leading spaces in a line. They are listed only to emphasize structures. You **DO** have to type a space between keywords in the program.

Long program lines: If a complete program line will not fit on one line, we will continue it onto the next line and add *//wrap line* at the end. You must type it as one continuous line.

Variable names, procedure names, and function names can be a combination of:

abcdefghijklmnopqrstuvwxyz
0123456789
'] [\ _

The apostrophe (') is often used. Note that the <left arrow> key in the upper left corner of the keyboard is also a valid character. COMAL 2.0 converts it into an underline (see the Fractal program for an example). If you see an underline in a program listing, type the <left arrow> key. The C64 and C128 computers use a <british pound> in place of the backslash (\).

Questions

K. Keller, Ladora, Iowa has 3 questions:
1) I want to convert *"Beekeeper"* to COMAL 0.14. It uses custom characters. How would it be done?

The font editor on page 7 allows you to redefine just a few characters. The modifed font is then saved to disk. Other programs can use the routines in LOAD'FONT/DEMO to access the new fonts. Both programs are on *Today Disk #12.*

2) I liked the picture captioned, *"He forgot to save"* on the front cover of *COMAL Today #6.* Does COMAL have an equivalent of BASIC's VERIFY?

COMAL 0.14 is only for disk SAVE & LOAD. The disk drive does an automatic verify. But 2.0 allows tape storage and thus includes VERIFY and allows you to verify on disk as well.

3) Does Machine Language code have to be POKEd in from within a COMAL program?

You can use POKEs or load directly from disk via the LOAD'OBJ routine on the back cover of COMAL Today #6. COMAL 2.0 allows you to use packages and the LINK command as well.

BECAUSE OF COMAL

I personally became interested in COMAL after a young cousin visiting from Denmark rather dramatically confirmed that COMAL was indeed the educational language of the Scandanavian nations. Because of COMAL we have 12 year olds writing very entertaining animation programs. They USE the *sprites* with an ease only dreamed of in BASIC. Keep up the good work. *COMAL Today* beats the other magazines hands down. - Doug Colpitts, John Norquay Elementary School ■

Adjust Circle

by David Stidolph

When drawing circles in COMAL 2.0, the CIRCLE command draws more of an oval than a true circle. This is because the individual pixels on the screen are taller than they are wide. The WINDOW command can overcome this by changing the screen scaling. Use the following program to determine what WINDOW command settings to use with your monitor. Run the program and press the cursor keys to adjust the circle so that it looks perfectly round. Write down the WINDOW setting displayed, and use it in your programs.

```
USE graphics
graphicscreen(0); window(0,319,0,199)
plottext(0,192,"Use CRSR keys to adjust WINDOW settings")
plottext(72,182,"Press the Q key to quit")
circle(160,100,50)
plottext(0,0,"window(0,319,0,199)")
x:=319; y:=199
LOOP
 REPEAT
  valid:=TRUE
  CASE KEY$ OF
  WHEN ""17"" // cursor down
   IF y>0 THEN y:-1
  WHEN ""145"" // cursor up
   IF y<32766 THEN y:+1
  WHEN ""157"" // cursor left
   IF x>0 THEN x:-1
  WHEN ""29"" // cursor right
   IF x<32766 THEN x:+1
  WHEN "q","Q"
   textscreen
   PAGE
   PRINT "Use the following commands to reset"
   PRINT "the graphics screen to draw circles"
   PRINT "like the one on the graphics screen."
   PRINT
   PRINT "USE graphics"
   END "window(0,",x,",0,",y,")"
  OTHERWISE
   valid:=FALSE
  ENDCASE
 UNTIL valid
 plottext(0,0,"window(0,"+STR$(x)+",0,"+STR$(y)+")   ")
 pencolor(-1); circle(160,100,50)
 window(0,x,0,y)
 pencolor(1); circle(160,100,50)
ENDLOOP■
```

Order Form

Name:_____ SUBSCRIBER NUMBER:_____ Apr. 1986
(required for reduced prices-except new subs)
Street:_____ Pay by check/MoneyOrder in US Dollars
Canada Postal US Dollar Money Order is OK
City/St/Zip:_____ VISA / MasterCard print card#/exp date:

VISA/MC #:_____exp date:_____Signature:_____

Qnty Price List/Subscriber price-Item Description (all disks Commodore 1541 format) Prices subject to change

SYSTEMS: *(two disks may be supplied as a double sided disk)*

[ ] _____ $98.95/$94.95 Deluxe Cartridge Pak (2 books/1 disk/1 cartridge)-(shipping add $4)
[ ] _____ **$19.95/$17.95 Programmers Paradise Pak (Fastloaded Comal 0.14/400 pgs info)(shipping add $2)**
[ ] _____ **$5/$5 option only with Paradise Pak - (includes Tutorial disk, Best of Disk, Auto Run Demos)**
[ ] _____ $350/$350 IBM Denmark PC COMAL (Danish manual/COMAL Handbook)-(shipping add $5)

SUBSCRIPTIONS:

[ ] _____ **COMAL Today newsletter-> How many issues? _____ Start with: 6 7 8 9 10 11 12<-Circle one**
($14.95 first 6; $2 each added issue; >>> Canada add $1 per issue; >>> overseas add $5 per issue)
[ ] _____ **$14.95/$12.95 COMAL Yesterday, first 4 issues COMAL Today spiral bound (ship add $2)**
[ ] _____ $3.95/$2.95 **COMAL Today backissue: circle issues wanted-> 5 6 7 8 9 10 11-(ship add $1 each)**
[ ] _____ **TODAY DISK subscription >>>>>>>>> How many disks? _____ Start with disk#_____**
($49.95/$35.95 for first 6 disks - $5 each added disk)-(**no extra shipping charge**)

BOOKS: (optional matching disks are $14.95/$4.95)-(>>>>**Canada shipping add $1 more per book**<<<<)

[ ] _____ **$19.95/$17.95 Introduction to Computer Programming (American 2.0 Tutorial)(ship add $3)**
[ ] _____ **$5.95 Introduction to Computer Programming Answers Book (ship add $1)(both due June 86)**
[ ] _____ $18.95/$16.95 COMAL Handbook - optional disk [___] (shipping add $3)
[ ] _____ $17.95/$15.95 Starting With COMAL (matching disk not available yet)-(shipping add $3)
[ ] _____ $19.95/$17.95 Foundations With COMAL - optional disk [___] (shipping add $3)
[ ] _____ $28.95/$26.95 Structured Programming With COMAL - optional disk [___] (shipping add $3)
[ ] _____ $20.95/$18.95 Beginning COMAL - optional disk [___] (shipping add $3)
[ ] _____ $17.95/$15.95 C64 Graphics With COMAL 0.14 - **optional disk** [___] (shipping add $3)
[ ] _____ $6.95/$4.95 COMAL From A To Z (part of Paradise Pak)-(shipping add $2)
[ ] _____ $14.95/$12.95 Captain COMAL Gets Organized (includes disk)-(shipping add $2)
[ ] _____ $6.95/$4.95 COMAL Workbook (perfect companion to Tutorial Disk)-(shipping add $2)
[ ] _____ $14.95/$12.95 COMAL Library of Functions & Procedures (includes disk)-(shipping add $2)
[ ] _____ $14.95/$12.95 Graphics Primer (includes disk) (for COMAL 0.14)-(shipping add $2)
[ ] _____ $6.95/$4.95 Cartridge Graphics & Sound (part of Deluxe Cartridge Pak)-(shipping add $2)
[ ] _____ $19.95/$17.95 COMAL 2.0 Packages (disk includes C64SYMB & Monitor)-(shipping add $2)
[ ] _____ **$19.95/$17.95 Packages Library (with disk)-(shipping add $2)**
[ ] _____ $19.95/$17.95 Cartridge Tutorial Binder (with disk)(part of Deluxe Cart Pak)(shipping add $3)
[ ] _____ $14.95/$12.95 COMAL Quick 0.14 (fastloaded) with Utility Disk #2 & book (shipping add $2)

DISKS:

[ ] _____ $94.05/$89.95 19 Disk Set (about 1000 programs for COMAL 0.14)-(shipping add $3)
[ ] _____ $14.95/$9.75 Best of COMAL 0.14 (new version - single side of disk)
[ ] _____ $14.95/$9.75 Auto RUN Demo and Tutorial Disk (perfect with COMAL Workbook)
[ ] _____ $14.95/$9.75 Bricks Tutorials (2 sided BEGINNERS disk)
[ ] _____ $14.95/$9.75 Utility Disk #1 for COMAL 0.14
[ ] _____ $14.95/$9.75 Today Disk-**Circle disks wanted: 1 2 3 4 5 6 7 8 9 10 11 12 (see sub above)**
[ ] _____ $10/$9.75 User Group Disks-**Circle disks wanted: 1 2 3 4 5 6 7 8 9 10 11**
[ ] _____ $29.95/$14.95 **Set of all Cartridge Demo Disks (includes #1 #2 #3 and #4)**
[ ] _____ **$14.95/$9.75 Specialty Disks: [__] Games [__] Typing(2.0 only) [__] Modem [__] Fonts**

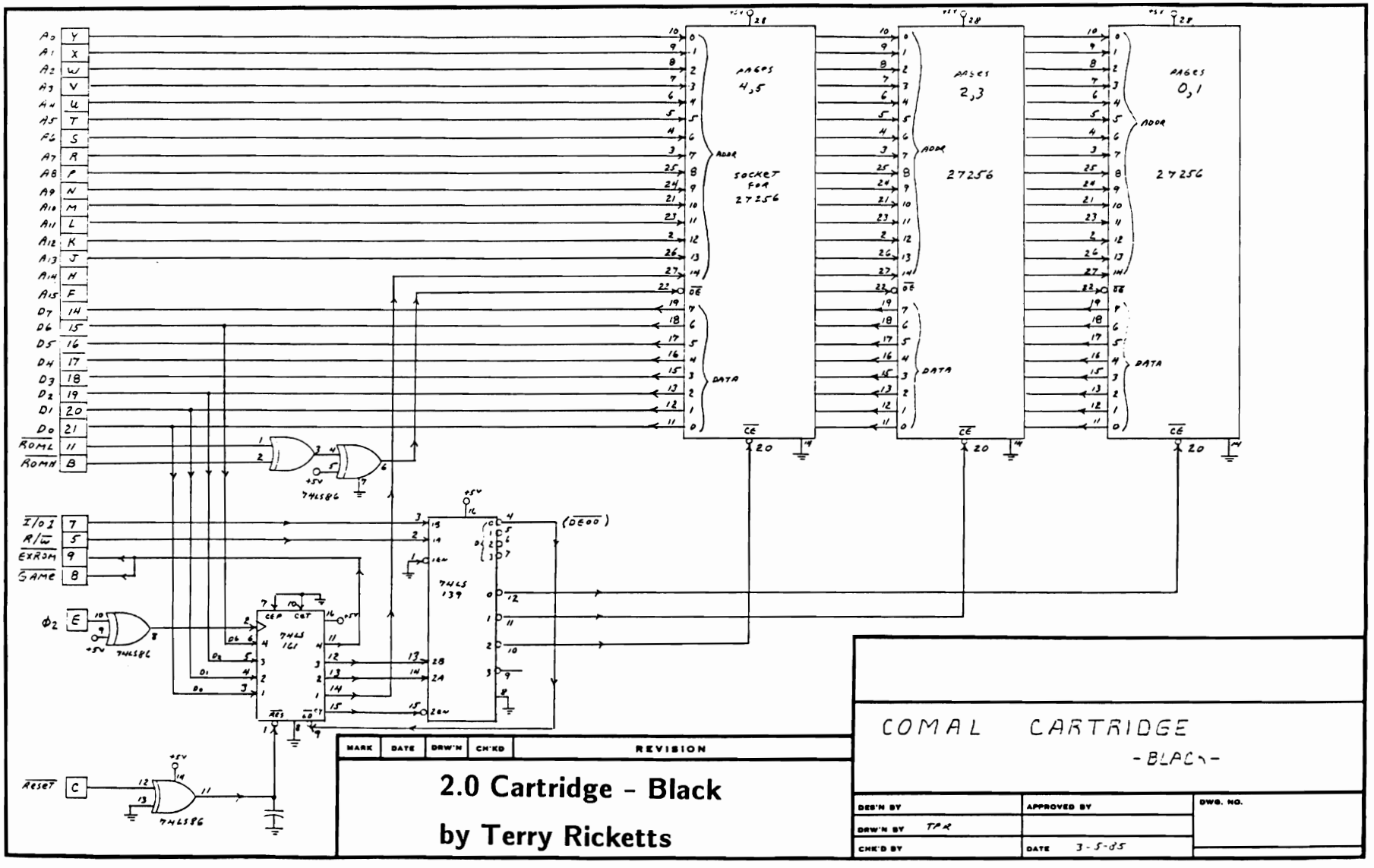
OTHER:

[ ] _____ OTHER: _____
[ ] _____ $3.95/$2.95 Keyboard Overlay for C64 COMAL 0.14 - Cheatsheet (shipping add $1)
[ ] _____ $49.95/$39.95 McPen Light Pen (with COMAL 2.0 Demo Disk)-(shipping add $3)
[ ] _____ $9.95/$5.95 Royal Blue COMALite Shirt-**circle ADULT size: S / M / L / XL)-(shipping add $2)**
[ ] _____ **$170.35 School COMAL Package - 12 different books and 12 different disks (shipping add $7.20)**

=====

Total_____ + _____ Shipping (minimum $2 per order) = **Total Paid** US$_____ (WI add 5% sales tax)
Mail To: COMAL Users Group USA, 6041 Monona Drive, Madison, WI 53716 or call 608-222-4432

Cartridge Schematic



Expand Past Maximum Capacity!

The Transactor

The Tech/News Journal For Commodore Computers

At better book stores everywhere! Or 6 issues delivered to your door for just $15.00
That's 29% off the newstand price! (Overseas $21 U.S. Air Mail $40 U.S.)
The Transactor. 500 Steeles Ave. Milton, Ontario. L9T 3P7. 416 878-8438
Also check out The Transactor Disk; every program from each issue, in order as they appear
and The Complete Commodore Inner Space Anthology; over 2.5 million characters of reference information exclusively.
Included are memory maps for BASIC, COMAL, CBM disk drives, BBS numbers, machine language charts, Kernel routine summaries, printer commands, recording formats, I/O port maps, port pinouts, audio control tables, video reference charts, keyword values, commands for common software packages, MLM and assembler commands, and there's more!
To us, expansion knows no limits!